LOGALI

Teoría

Tipos de Entidades y Servicios

ABAP Cloud – Modelado con CDS

Contenido

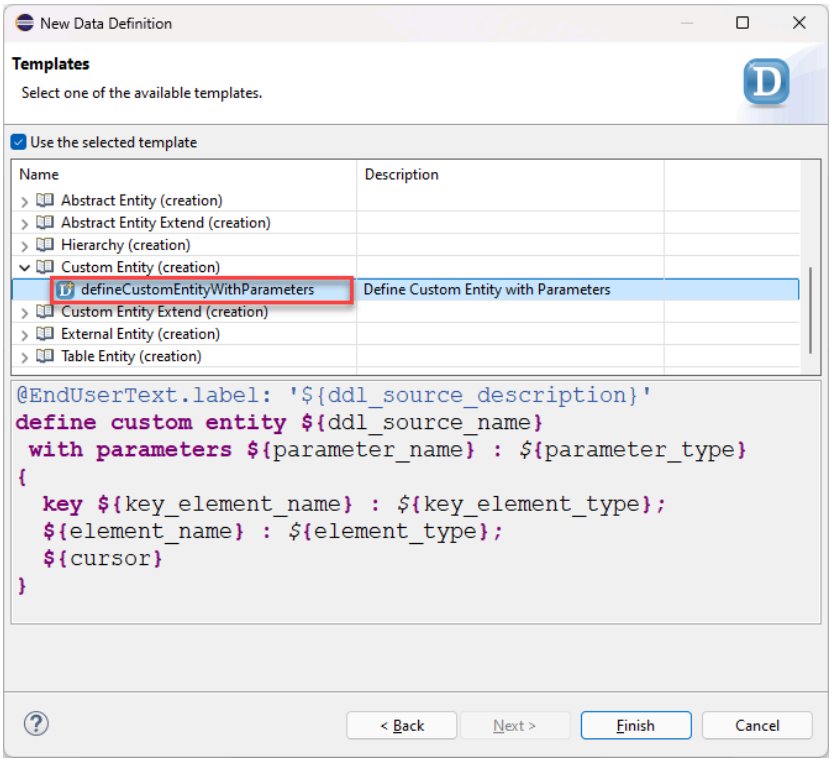
4. Tipos de Entidades y Servicios

4.1. Entidad personalizada - Custom Entity

Una entidad personalizada es una estructura de datos definida por el usuario en el sistema ABAP que no tiene una tabla base en la base de datos. Tiene un comportamiento similar a las estructuras del diccionario de datos para ser utilizadas para la creación de tablas internas para consultar datos de una vista cds o una tabla de base de datos. Estas entidades son sumamente útiles cuando necesitas manejar datos específicos que no están cubiertos por las tablas estándar del sistema. Una entidad personalizada se define dentro de un Core Data Services (CDS) view, permitiendo una mayor flexibilidad y adaptación a las necesidades del negocio. Además, las entidades personalizadas pueden ser consultadas y manipuladas utilizando Open SQL, lo que facilita su integración con otras partes del sistema.

Una vez definida la entidad personalizada y configurado el service binding se pueden realizar peticiones OData a la entidad. El servicio OData permite que los datos de la entidad personalizada sean accesibles desde aplicaciones externas mediante solicitudes HTTP.

Para crear una entidad personalizada, se realiza a través de la carpeta de proyecto luego New en la opción **Other ABAP Repository Object** ubicar la carpeta **Business Services** y luego **Service Definition** y seleccionar la plantilla **defineCustomEntityWithParameters** que se encuentra en la carpeta **Custom Entity (creation).**



Sintaxis básica para la creación de una entidad personalizada:

```
@EndUserText.label: 'Custom Entity Name'
@ObjectModel.query.implementedBy: 'ABAP:zclass_name'
define custom entity entity_name
{
    element_name     : data_type;
...
}
```

Sintaxis básica para la creación de la clase que realizará los query utilizando la entidad personalizada:

```
CLASS zclass_name DEFINITION
 PUBLIC
 FINAL
 CREATE PUBLIC .
 PUBLIC SECTION.
   INTERFACES if_rap_query_provider .
 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.
```

```
CLASS zcl_entity_test_100437 IMPLEMENTATION.
 METHOD if_rap_query_provider~select.
logic
 ENDMETHOD.
ENDCLASS.
```

Ejemplo de una query de una entidad personalizada en una clase:

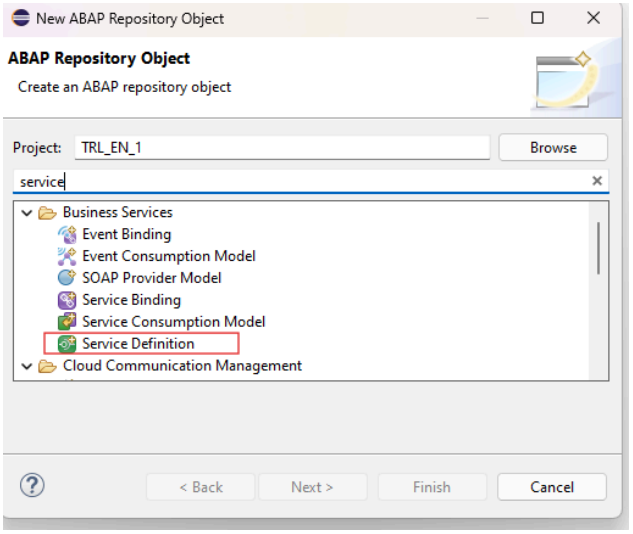
```
CLASS zclass_nameIMPLEMENTATION.
 METHOD if_rap_query_provider~select.
  DATA lt_results TYPE TABLE OF zentity_name.
  TRY.
    IF io_request->is_data_requested( ).
     DATA(lv_top) = io_request->get_paging( )->get_page_size( ).
     DATA(lv_skip) = io_request->get_paging( )->get_offset( ).
     SELECT FROM table_name
     FIELDS component,
               …
     ORDER BY component ASCENDING
     INTO TABLE @lt_results
     OFFSET @lv_skip
     UP TO @lv_top ROWS.
     IF sy-subrc EQ 0.
       io_response->set_total_number_of_records( lines( lt_results )
).
       io_response->set_data( lt_results ).
     ENDIF.
    ENDIF.
   CATCH cx_rap_query_response_set_twic INTO DATA(lx_error).
  ENDTRY.
 ENDMETHOD.
ENDCLASS.
```

4.2. Service Definition

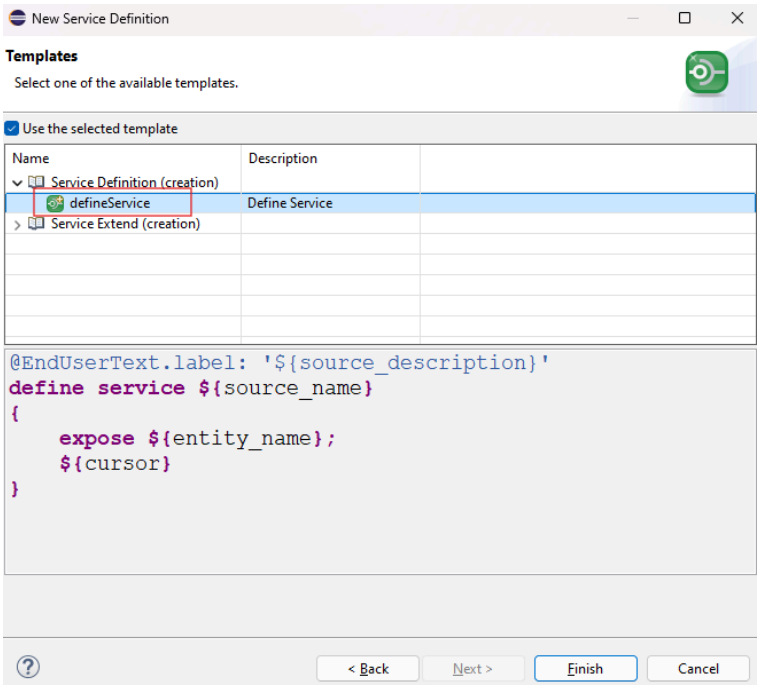
Una definición de servicio especifica qué entidades y operaciones están disponibles a través de un servicio. Esto actúa como una interfaz pública para el servicio, exponiendo sólo aquellas entidades

y operaciones que se desean compartir con clientes externos. Las definiciones de servicio son cruciales en arquitecturas basadas en servicios, ya que permiten la comunicación entre diferentes componentes del sistema de una manera controlada y segura.

Para la creación de un Service definition se hace a través de la ventana New luego **Other ABAP Repository Object**, se ubica la carpeta **Business Services** y se selecciona la opción **Service Definition.**



Es importante que al momento de asignar la orden de transporte se seleccione la opción de next para luego seleccionar la plantilla necesaria **defineService** que por defecto vendrá seleccionada.



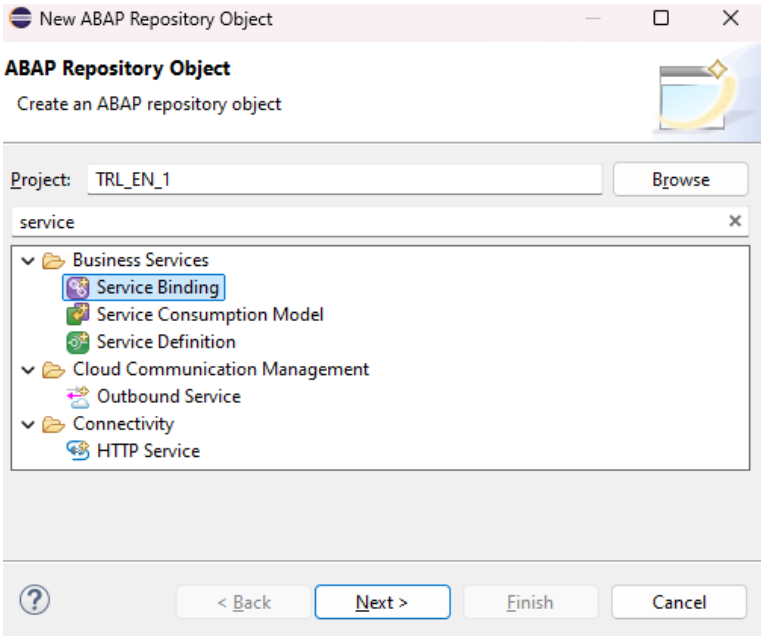
Sintaxis de un servicio:

```
@EndUserText.label: 'Basic Service'
define service zservice_name
{
  expose entity_name as AliasName;
  expose entity_name;
 ...
}
```

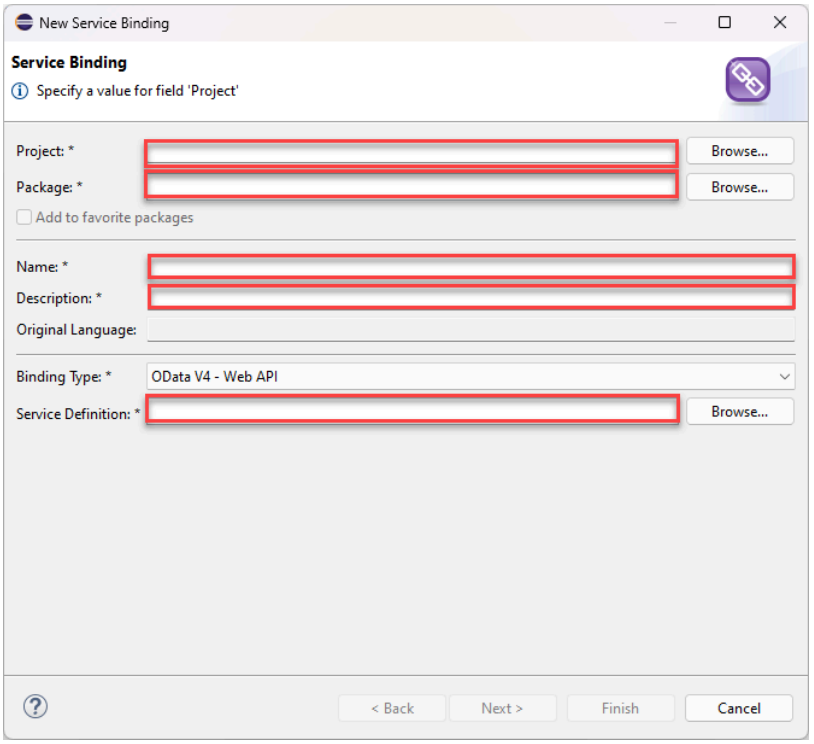
4.3. Service Binding

El service binding es un componente esencial en el desarrollo de aplicaciones en ABAP, ya que permite vincular el tipo de servicio y publicar el endpoint que da acceso a las peticiones realizadas desde fuera del sistema. Este proceso es crucial para exponer las entidades personalizadas y otros servicios a través de OData, facilitando la integración y el acceso a los datos desde aplicaciones externas.

Para crear un service binding se puede hacer a través del paquete con la opción **Other ABAP Repository Object** luego seleccionar **Business Service** y por último **Service Binding.** También directamente desde la carpeta **Business Service** luego de haber creado cualquier entidad o service binding dentro de la carpeta del paquete.

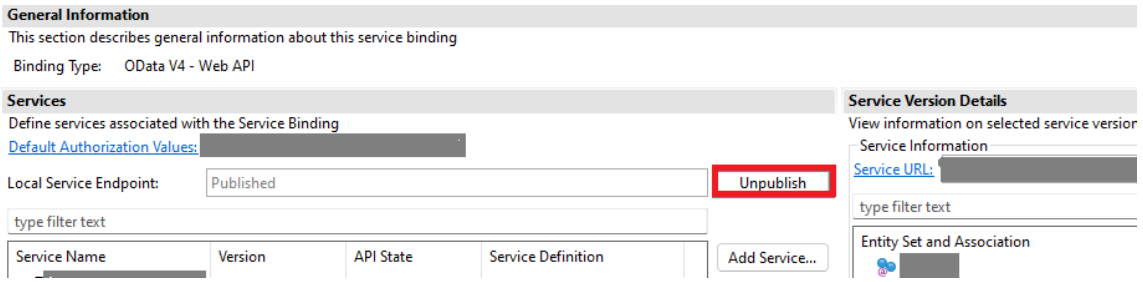


Se debe colocar un nombre, una descripción, en la opción **binding type:** seleccionar OData V4 Web Api y colocar el nombre de la entidad personalizada a la cual se le necesita crear un servicio Odata.



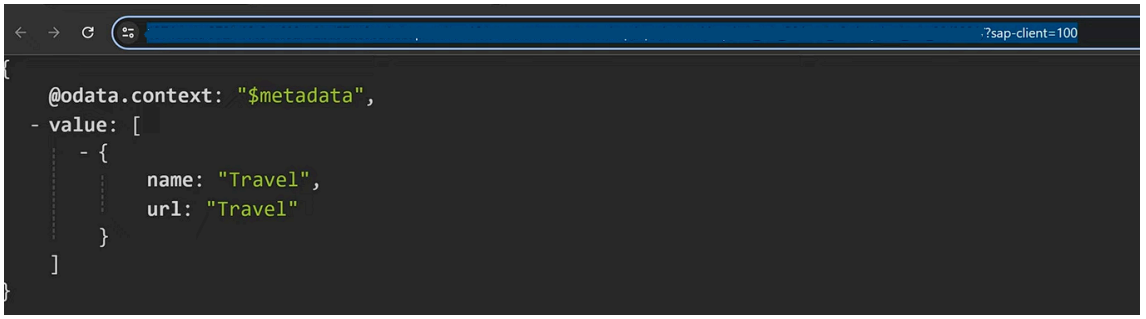
Es necesario que al momento de crear el servicio activar el service binding para poder presionar el botón **Publish** para generar el url

donde se podrán consultar datos por medio de la entidad y la clase que está referenciada datos en una base de datos.



Al presionar el enunciado **Service URL:**

Redireccionará al navegador en donde si previamente se ha iniciado sesión en el BTP no pedirá las credenciales para ingresar. Mostrando los resultados de la consulta asociada a la clase especificada en la entidad.



Algunas operaciones que se pueden realizar en el URL debido a que en la clase de ejemplo del tema **Entidad personalizada - Custom Entity** que fueron agregadas en la consulta serían las siguientes:

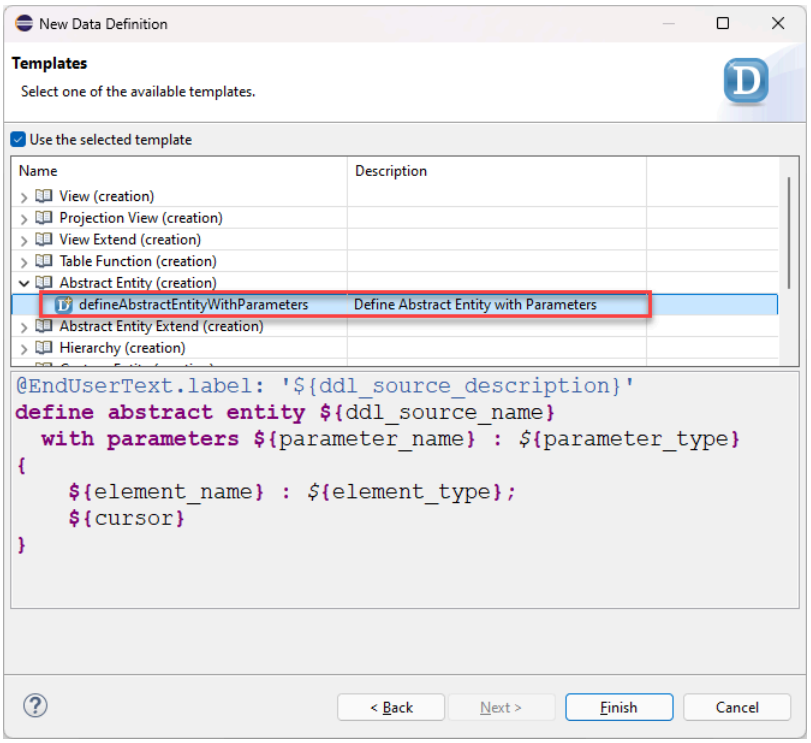
- **url+name_entity_set_association?sap-client=100.** Para consultar todos los registros de la tabla con el nombre de la asociación. Ejemplo: URL+Travel?sap-client=100. El símbolo más es solo para señalar que antes del nombre de la asociación estará todo el URL completo
- **url+name_entity_set_association?sap-client=100&$top = number_of_records.** Para limitar los resultados de búsqueda colocados a través de un número en la sección number_of_records. Ejemplo: URL+Travel?sap-client=100&$top=10. El símbolo más es solo para señalar que antes del nombre de la asociación estará todo el URL completo

4.4. Entidad abstracta – Abstract Entity

Una entidad abstracta en ABAP es una estructura de datos que no persiste en la base de datos. Estas entidades se utilizan principalmente para definir estructuras de datos complejas que pueden ser utilizadas en cálculos temporales, transformaciones de datos, o para representar datos en modelos jerárquicos sin necesidad de almacenamiento permanente. Se utilizan en escenarios donde los datos no necesitan ser almacenados de manera persistente, sino que se utilizan para cálculos temporales, transformaciones o para representar estructuras de datos jerárquicas de manera eficiente. Estas entidades son ideales para modelar datos complejos que necesitan ser procesados rápidamente en memoria.

Para crear una entidad abstracta se hace el mismo procedimiento para crear una vista CDS se realiza a través de la carpeta de proyecto

luego New en la opción **Other ABAP Repository Object** ubicar la carpeta **Business Services** y luego **Data Definition**. Con la diferencia que la plantilla que debe ser seleccionada es la plantilla **defineAbstractEntityWithParameters** que se encuentra en la carpeta **Abstract Entity (creation).**



Sintaxis:

```
@EndUserText.label: 'Abstract Entity'
define abstract entity abstract_entity_name
{
  element_name : element_type;
...
}
```

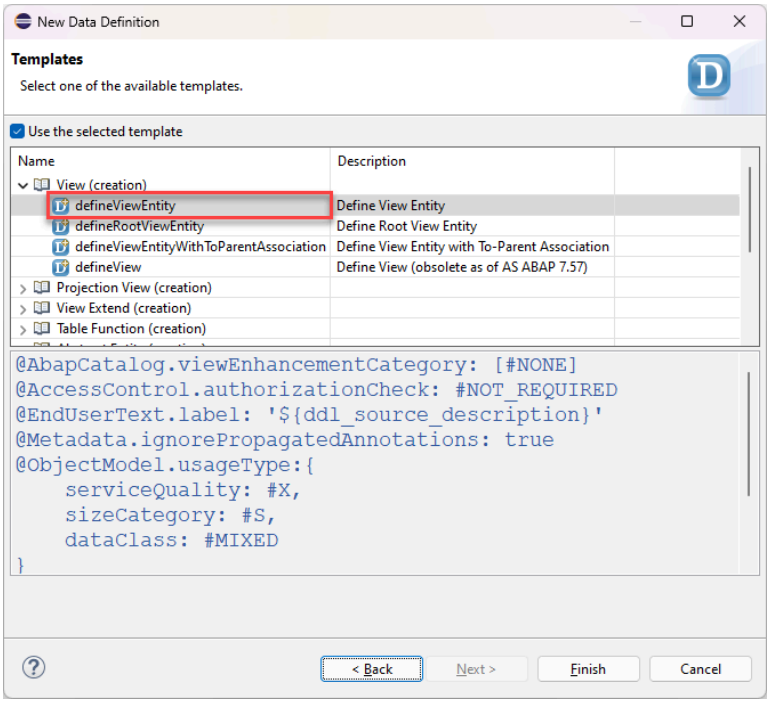
La manera en la que definimos una tabla interna a partir de una entidad abstracta se realiza de la siguiente manera.

```
DATA lt_itab TYPE TABLE OF abstract_entity_name.
```

4.5. CDS para modelo Jerárquico

Permiten la creación y gestión de modelos jerárquicos de datos. Estos modelos se utilizan para representar relaciones estructurales complejas. La implementación de CDS para modelos jerárquicos implica la definición de una tabla de persistencia que incluye identificadores únicos para cada registro y referencias a sus respectivas jerarquías. Utilizando vistas CDS, se proyectan las columnas necesarias y se establecen asociaciones a la misma tabla de base de datos que reflejan estas relaciones jerárquicas. Este enfoque permite consultar y navegar eficientemente a través de los datos jerárquicos, optimizando el rendimiento y garantizando la integridad de los datos.

Para crear una entidad o vista CDS necesaria para la asociación parent child, se realiza el mismo procedimiento para crear una entidad o vista CDS 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).**



Es importante mencionar que para realizar una asociación en un modelo jerárquico utilizando CDS ABAP, es fundamental consultar la misma entidad, lo que permite construir una jerarquía de datos dentro de una única estructura. La asociación debe realizarse sobre un campo clave y otro campo de la misma entidad. Es importante que la tabla de base de datos utilizada como fuente de datos contenga registros donde los valores del campo clave coincidan con los del otro campo. Además, ambos campos deben tener el mismo tipo de dato y longitud para que la relación sea válida y consistente.

Sintaxis

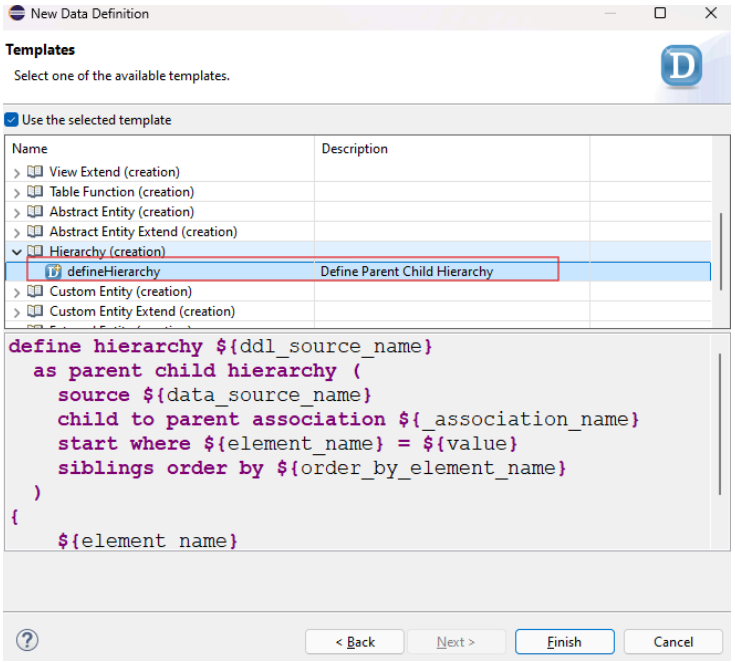
```
define view entity entity_name
 as select from table_name as AliasName
   association [min..max] to entity_name as _AliasName on
_AliasName.component_name2 = $projection.ComponentName
...
{
        component_name,
        component_name2,
       ...
        _AliasName
}
```

4.6. Entidad de Jerarquía – Hierarchy

Una entidad de jerarquía es un tipo de estructura de datos utilizada para representar relaciones jerárquicas entre elementos dentro de un sistema. En el contexto de ABAP y CDS (Core Data Services), las jerarquías permiten modelar y gestionar relaciones de tipo padre-hijo, donde una entidad (padre) puede tener múltiples entidades subordinadas (hijos), y estas a su vez pueden tener sus propias entidades subordinadas. Esto es útil para organizar datos de manera que reflejen la estructura real del negocio. Se utiliza para organizar y estructurar datos de manera jerárquica, facilitando el acceso y la administración de estos datos. Las jerarquías son especialmente útiles en aplicaciones que manejan datos complejos y estructurados, como organigramas, catálogos de productos, estructuras de proyectos, y más. Permiten realizar operaciones y

consultas que aprovechan estas relaciones jerárquicas para obtener información de manera más eficiente.

Para crear una entidad de jerarquía se hace el mismo procedimiento para crear una vista CDS se realiza a través de la carpeta de proyecto luego New en la opción **Other ABAP Repository Object** ubicar la carpeta **Business Services** y luego **Data Definition**. Con la diferencia que la plantilla que debe ser seleccionada es la plantilla **defineHierarchy** que se encuentra en la carpeta **Hierarchy (creation).**



En la definición de la jerarquía se pueden configurar ciertas características a través de las sentencias:

- **siblings order by order_by_element_name**: Ordena los elementos hermanos de acuerdo con order_by_element_name.
- **multiple parents allowed**: Permite que un nodo tenga múltiples padres.
- **orphan ignore**: Ignora los nodos huérfanos en la jerarquía.
- **cycles breakup**: Rompe ciclos en la jerarquía para evitar bucles infinitos.
- **generate spantree**: Genera un árbol de alcance (span tree). Asegurar que no hay múltiples padres por lo que las no es

posible utilizar esta sentencia en conjunto con **multiple parents allowed, orphan ignore** y **cycles breakup.**

- **cache on**: Activa la caché para las consultas, mejorando el rendimiento.

Es posible la proyección de elementos o componentes dentro del cuerpo de la jerarquía para obtener una columna al momento de consultar información con las características de la jerarquía. Esto a través de la siguiente manera:

- **element_name as AliasName**: El nombre del elemento o componente normal que se proyectará.
- **$node.node_id as AliasName**: Proyecta el ID del nodo.
- **$node.hierarchy_is_cycle as AliasName**: Indica si el nodo es parte de un ciclo.
- **$node.hierarchy_is_orphan as AliasName**: Indica si el nodo es huérfano.
- **$node.hierarchy_level as AliasName**: Proyecta el nivel jerárquico del nodo.
- **$node.hierarchy_parent_rank as AliasName**: Proyecta el rango del padre en la jerarquía.
- **$node.hierarchy_rank as AliasName**: Proyecta el rango del nodo en la jerarquía.
- **$node.hierarchy_tree_size as AliasName**: Proyecta el tamaño del árbol desde el nodo

Sintaxis:

```
@AccessControl.authorizationCheck: #NOT_ALLOWED

define hierarchy cds_name
  with parameters
 as parent child hierarchy (
  source cds_name2
  child to parent association _AssociationName
  start where element_name = value
  siblings order by order_by_element_name
  multiple parents allowed
  orphan ignore
```

```
  cycles breakup
  generate spantree
  cache on
 )
{
  element_name
  $node.node_id
  $node.hierarchy_is_cycle
  $node.hierarchy_is_orphan
  $node.hierarchy_level
  $node.hierarchy_parent_rank
  $node.hierarchy_rank
  $node.hierarchy_tree_size
}
```

La anotación **@AccessControl.authorizationCheck:** es fundamental para evitar que los datos sean mostrados incorrectamente debido a restricciones de acceso. En modelos jerárquicos y en las entidades de jerarquía, es crucial evitar que estas restricciones distorsionen la representación completa de la jerarquía. Utilizando en ambos casos la anotación:

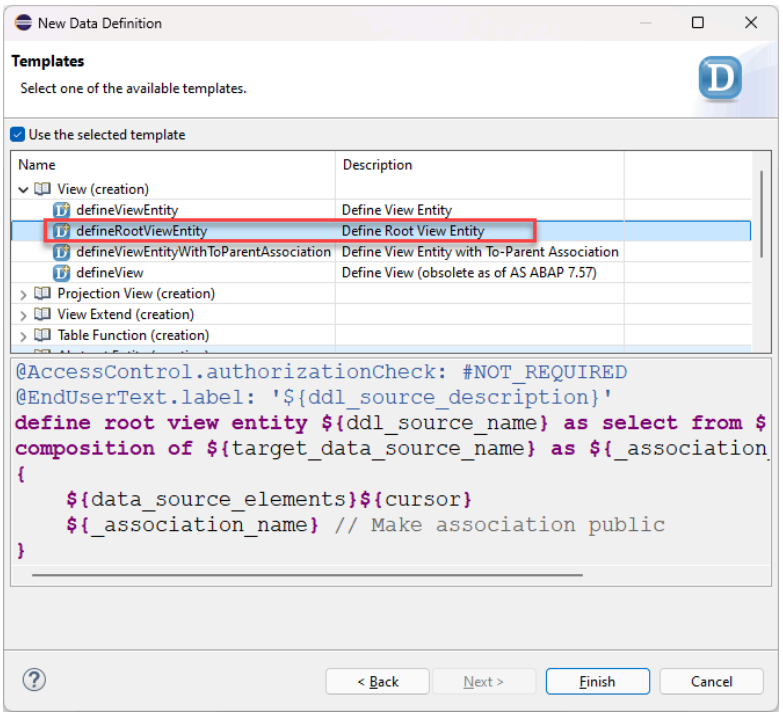
```
@AccessControl.authorizationCheck: #NOT_ALLOWED
```

4.7. Entidad raíz – Define Root Entity

Esta entidad actúa como el nodo principal en la jerarquía de datos y es el punto de entrada para todas las relaciones de negocio en una aplicación RAP. La entidad raíz facilita la gestión centralizada de datos complejos, estableciendo relaciones de composición y jerarquía con otras entidades del sistema. Su correcta implementación asegura la integridad y eficiencia en la manipulación y acceso a los datos del negocio.

Para crear una entidad raíz se hace el mismo procedimiento para crear una vista CDS se realiza a través de la carpeta de proyecto luego New en la opción **Other ABAP Repository Object** ubicar la carpeta **Business Services** y luego **Data Definition**. Con la diferencia que la plantilla que debe ser seleccionada es la plantilla

defineRootViewEntity que se encuentra en la carpeta **View (creation).**



Sintaxis:

```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'CDS Root Entity'
define root view entity entity_name
 as select from data_source_name
 //composition of target_data_source_name as _AssociationName
   association [min..max] to entity_name as _AliasName on
_AliasName.ComponentName = $projection.ComponentName
...
{
 _AssociationName // Make association public
}
```

4.8. Asociación - Parent Child

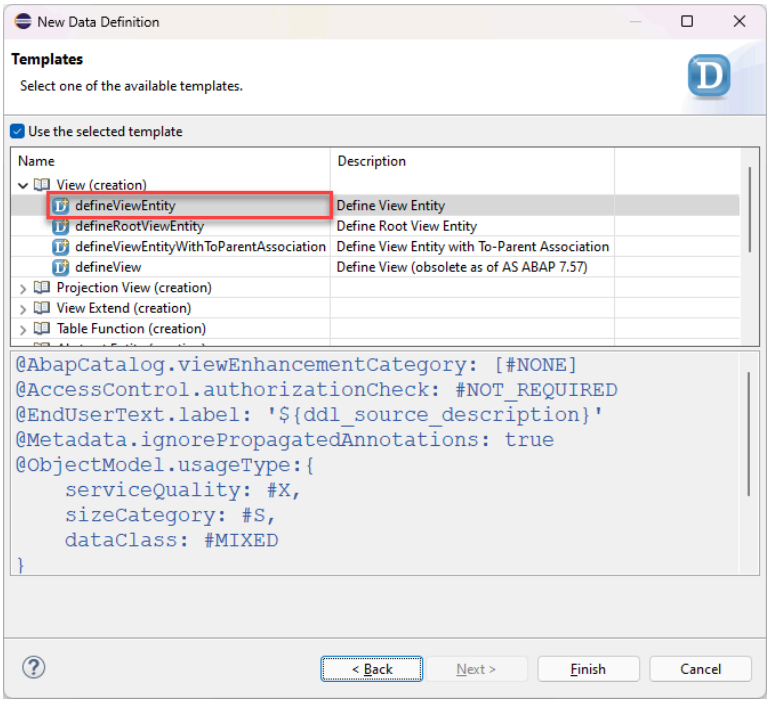
Es una relación clave dentro del modelo de programación RAP (ABAP RESTful Application Programming) que define como una entidad hija (Child) se relaciona y navega hacia su entidad padre (Parent). Esta relación asegura la integridad y coherencia de los datos, permitiendo que las entidades hijas sólo existan dentro del

contexto de su entidad padre. La correcta configuración y publicación de estas asociaciones es fundamental para mantener la estructura y la navegación eficientes dentro del modelo de datos, asegurando que las entidades hijas puedan acceder a los datos del padre y viceversa de manera efectiva. Se utiliza para modelar relaciones jerárquicas entre entidades, permitiendo consultas que naveguen a través de estas relaciones y operaciones conjuntas en las entidades relacionadas. Esto es esencial para aplicaciones que manejan datos jerárquicos como estructuras organizacionales, catálogos de productos, entre otros.

Puntos importantes a tomar en cuenta:

- **Relaciones únicas con una sola entidad padre:** Las entidades hijas solo pueden tener una relación de tipo padre. No es posible que una entidad hija tenga múltiples padres. Esto asegura la individualidad y coherencia de las relaciones jerárquicas en el modelo de datos.
- **Entidad Raíz y Asociaciones:** La entidad raíz (root entity) no puede ser hija de otra entidad. Cada entidad raíz debe ser única y no debe tener asociaciones padre, manteniendo su rol como el nodo principal en la jerarquía de datos. Sin embargo, puede establecer asociaciones con otras entidades, a través de relaciones con otra cardinalidad especificada. El padre agrega la cardinalidad de los posibles hijos que se pueden esperar como registros desde las entidades de tipo hijas.
- **Uso en Consultas y Navegación:** Una vez definidas y publicadas las asociaciones, se pueden realizar consultas que naveguen a través de estas relaciones, proporcionando acceso estructurado y eficiente a los datos relacionados.
- **Publicar una asociación del padre obligatoriamente**: Si una asociación no se publica correctamente, el sistema puede generar mensajes de advertencia o errores. Es crucial manejar estas publicaciones para asegurar que las relaciones se integren adecuadamente en el modelo de datos.
- **Es opcional publicar cualquier otra asociación**: Es posible tener relaciones con otras entidades dentro de la entidad, y no es obligatorio tener que publicar la asociación en los componentes de la entidad.

Para crear una entidad o vista CDS necesaria para la asociación parent child, se realiza el mismo procedimiento para crear una entidad o vista CDS 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: [#NONE]
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'CDS Association Parent Child'
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{
  serviceQuality: #X,
  sizeCategory: #S,
  dataClass: #MIXED
}
define view entity entity_name
 as select from
```

```
  association to parent entity_name as _AliasParentName on
_AliasParentName.component_name                          =
$projection.ComponentName
  association [min..max] to entity_name as _AliasName on
_AliasName.component_name = $projection.ComponentName
...
{
   component_name as AliasName,
   _AliasParentName , //Mandatory
   _AliasName //Optional
}
```

4.9. Composición – Parent Child

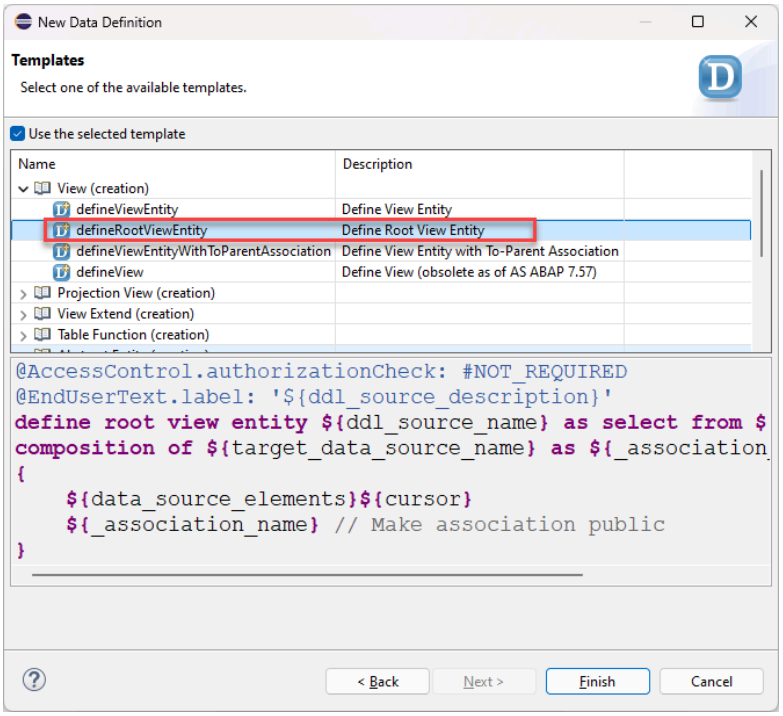
Es una relación fundamental en el modelo de programación RAP (ABAP RESTful Application Programming) que establece dependencias entre entidades relacionadas. Esta relación asegura que la entidad hija no puede existir sin la entidad padre, garantizando la integridad referencial de los datos. La entidad raíz define la relación de composición y cardinalidad, permitiendo una navegación y gestión eficiente de datos jerárquicos dentro del sistema. La correcta implementación de esta relación es esencial para mantener la coherencia y confiabilidad de los datos en aplicaciones empresariales.

Puntos importantes a tomar en cuenta:

- **Dependencia del Hijo al Padre:** En un modelo RAP, las entidades hijas no deberían existir sin la entidad padre. Esto asegura que si el padre es eliminado, las entidades hijas también se eliminan automáticamente. Esta dependencia es crucial para mantener la integridad referencial de los datos.
- **Publicar una asociación del padre obligatoriamente:** Si una asociación no se publica correctamente, el sistema puede generar mensajes de advertencia o errores. Es crucial manejar estas publicaciones para asegurar que las relaciones se integren adecuadamente en el modelo de datos.

- **Navegación en el Modelo:** Una vez configurada la relación de composición, se puede navegar entre las entidades padre e hija utilizando las asociaciones definidas.
- **Cardinalidad:** La relación de composición debe tener una cardinalidad correctamente definida. Por defecto, se asigna una cardinalidad de cero a uno [0..1], pero en muchos casos, la relación puede ser de cero a muchos. Esto se configura según la estructura de las claves en las entidades.
- **Relaciones de Clave:** La entidad padre y la entidad hija pueden tener diferentes estructuras de clave, lo que afecta la cardinalidad de la relación de composición.
- **Creación de registros:** Para crear una entidad hija, primero debe existir la entidad padre. Esta relación de dependencia debe ser respetada en todas las operaciones CRUD (Create, Read, Update, Delete).
- **Eliminación de registros:** Si se elimina un registro en la entidad padre, los registros en las entidades hijas relacionadas deben ser eliminadas automáticamente para mantener la coherencia de los datos. Esto refleja una relación empresarial donde, por ejemplo, no puede haber posiciones en una factura sin tener la factura en sí.

Para crear una entidad raíz se hace el mismo procedimiento para crear una vista CDS se realiza a través de la carpeta de proyecto luego New en la opción **Other ABAP Repository Object** ubicar la carpeta **Business Services** y luego **Data Definition**. Con la diferencia que la plantilla que debe ser seleccionada es la plantilla **defineRootViewEntity** que se encuentra en la carpeta **View (creation).**



Sintaxis:

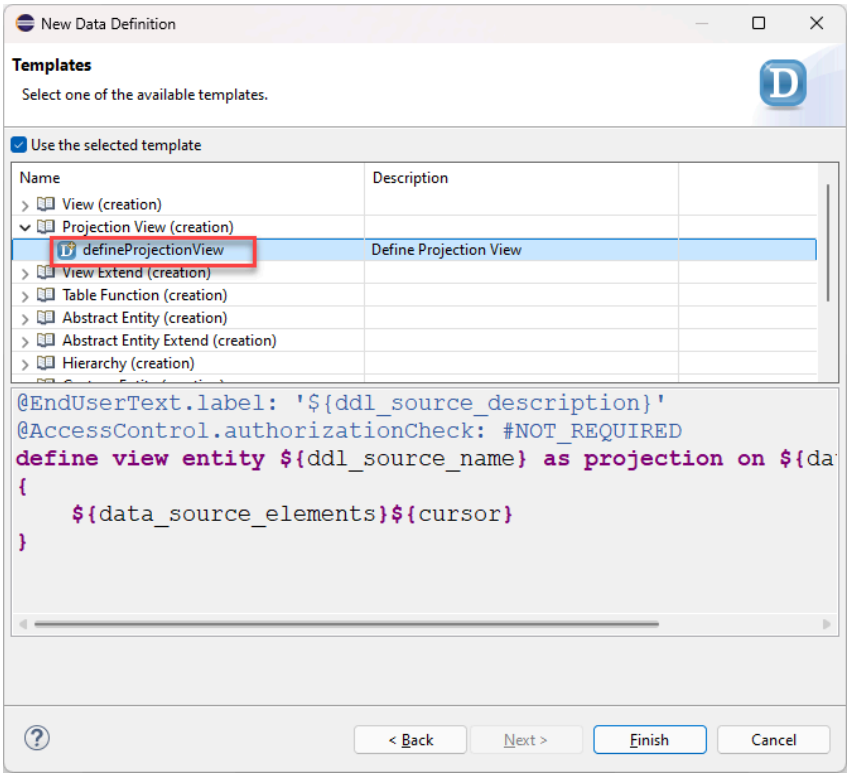
```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'CDS Root Entity'
define root view entity entity_name
 as select from data_source_name
   composition [min..max] of target_data_source_name as _AssociationName
   association [min..max] to entity_name as _AliasName on _AliasName.ComponentName = $projection.ComponentName
…
{
 _AssociationName //Mandatory
}
```

4.10. Proyección – Contrato Transaccional Interfaz

Es una técnica utilizada en el modelo RAP (ABAP RESTful Application Programming) que permite definir y gestionar los elementos de datos proyectados desde entidades base a través de una capa de interfaz. Esta técnica facilita la separación entre la lógica de negocio y la presentación de datos, lo que permite la creación de APIs y aplicaciones con acceso controlado a los datos.

Una proyección de contrato transaccional interfaz define una vista que expone solo las columnas relevantes para una interfaz transaccional específica. Este enfoque permite optimizar el acceso a los datos al proporcionar sólo la información necesaria para una determinada operación o consulta. Las proyecciones son esenciales en entornos donde la eficiencia y la seguridad de los datos son primordiales, ya que limitan la exposición de los datos sólo a aquellos campos que son necesarios para la transacción.

Para crear una proyección se realiza el mismo procedimiento para crear una entidad o vista CDS 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 **defineProjectionView** que se encuentra en la carpeta **Projection View (creation).**



Para la creación de proyección se define utilizando **define view entity**. Aunque siempre que se proyecte una entidad raíz(root) se deben agregar **root** en la declaración (**define root view entity**). En combinación con **as projection on** para especificar la entidad con los elementos que se van a proyectar. Esto asegura que solo los datos

necesarios se exponen a través de la interfaz, manteniendo la seguridad y la eficiencia.

Tipos de Contratos de Proyección:

En este tema se estará estudiando sólo el tipo de contrato **transactional_interface.** Aunque los contratos que se utilizan para las proyecciones son los **transactional_interface** y **transactional_query** para definir qué elementos se proyectan y cómo se accede a ellos, asegurando una estructura de datos clara y coherente:

- **Transaction_Interface:** Este tipo de contrato se utiliza para interfaces transaccionales que permiten la extensión del modelo con nuevas APIs y funcionalidades. Es ideal para operaciones CRUD (Create, Read, Update, Delete) en el contexto de objetos de negocio.
- **Transaction_Query:** Se utiliza para consultas transaccionales que proporcionan acceso a datos específicos sin modificar el estado del objeto de negocio. Es adecuado para reportes y análisis de datos.

Los siguientes tipos de contratos no se utilizan para las proyecciones utilizadas para RAP. Pero si definiciones de la capa analítica:

- **Analytical_Query:** Se utiliza para consultas analíticas que manejan grandes volúmenes de datos y proporcionan vistas agregadas y análisis detallados.
- **SQL_Query:** Este contrato permite realizar consultas SQL directamente desde el motor ABAP al motor HANA. Sin embargo, no es adecuado para objetos de negocio RAP, ya que está limitado a modelos de solo lectura.

Sintaxis:

```
@EndUserText.label: 'CDS Interface Projection'
@AccessControl.authorizationCheck: #NOT_REQUIRED
define root view entity entity_name
 provider contract transactional_interface
```

```
 as projection on data_source_name
{
component_name
…
 }
```

4.11. Redireccionamiento

El redireccionamiento es una técnica que permite redirigir la llamada de una entidad de proyección padre a hija, lo que facilita la gestión de cambios en la estructura de datos o en la lógica de negocio sin necesidad de modificar el cliente que realiza la llamada. Esto se logra configurando las entidades de las proyecciones para que cualquier solicitud dirigida a una entidad sea redirigida automáticamente a otra.

Si se realiza la proyección a una entidad **root** con una composición que tiene asociada una asociación del tipo parent child, y al establecer un tipo de contrato como **transactional_interface** ó **transactional_query** en la proyección (**define root view entity**), se clasificaría como una proyección padre. Por lo que se debe crear una proyección adicional a la entidad definida en la composición de la entidad root (la asociación del tipo parent child) sin ningún tipo de contrato. Esta proyección adicional se clasifica como una proyección hija. Para la creación de esta entidad hija se utiliza la misma plantilla de la entidad padre **defineProjectionView.**

Estas definiciones implican que la entidad de proyección raíz (**root view entity**) y la entidad hija, que tiene una relación con la entidad padre a través de composiciones.

Se pueden especificar redireccionamientos entre la publicación de las asociaciones entre el padre y el hijo agregando en dichas publicaciones lo siguiente:

Proyección raíz o padre:

_AssociationName**: redirected to composition child** child_projection_name.

Proyección hija:

_AssociationName**: redirected to parent** root_projection_name.

Ejemplo:

Proyección raíz o padre:

```
@EndUserText.label: 'CDS Interface Projection'
@AccessControl.authorizationCheck: #NOT_REQUIRED
define root view entity entity_name
 provider contract transactional_interface
 as projection on entity_root_name
{
component_name
…
_AssociationName: redirected to composition child
child_projection_name.

 }
```

Proyección hija:

```
@EndUserText.label: 'CDS Interface Projection'
@AccessControl.authorizationCheck: #NOT_REQUIRED
define view entity entity_name
 as projection on parent_child_association
{
component_name
…
_AssociationName: redirected to parent root_projection_name.

 }
```

Entidad root:

```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'CDS Root Entity'
define root view entity entity_root_name
 as select from data_source_name
   composition [min..max] of parent_child_association as
_AssociationName
```

```
  association [min..max] to entity_name as _AliasName on
_AliasName.ComponentName = $projection.ComponentName
…
{
 _AssociationName //Mandatory
}
```

Entidad con asociación Parent Child

```
@AbapCatalog.viewEnhancementCategory: [#NONE]
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'CDS Association Parent Child'
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{
  serviceQuality: #X,
  sizeCategory: #S,
  dataClass: #MIXED
}
define view entity entity_name
 as select from
  association to parent entity_name as _AliasParentName on
_AliasParentName.component_name =
$projection.ComponentName
  association [min..max] to entity_name as _AliasName on
_AliasName.component_name = $projection.ComponentName
...
{
   component_name as AliasName,
   _AliasParentName , //Mandatory
   _AliasName //Optional
}
```

4.12. Proyección – Contrato Transaccional Query

Una proyección de contrato **transaccional_query** es similar a la interfaz transaccional, pero está orientada a consultas específicas que necesitan un conjunto particular de datos. Estas proyecciones permiten optimizar las consultas al proporcionar solo las columnas y

los registros que son relevantes para la consulta en cuestión, mejorando la eficiencia y el rendimiento de las consultas de datos.

Sintaxis:

```
@EndUserText.label: 'CDS Interface Projection'
@AccessControl.authorizationCheck: #NOT_REQUIRED
define root view entity entity_name
 provider contract transactional_query
 as projection on entity_root_name
{
component_name
…
_AssociationName: redirected to composition child child_projection_name.

 }
```