LOGALI

Teoría

Modelo de Datos

ABAP RESTful – Arquitectura Cloud

Contenido

1. Modelo de Datos

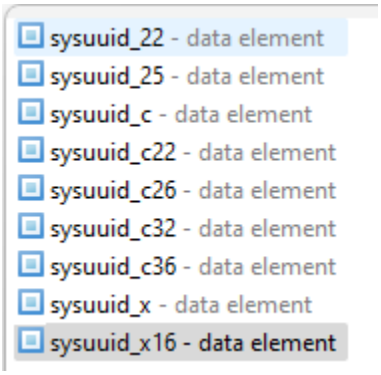
1.1. Persistencia Activa

El desarrollo de aplicaciones RAP en SAP BTP es un proceso que abarca desde la configuración del entorno de desarrollo hasta la implementación y activación de tablas de persistencia. La utilización de elementos de datos existentes, la implementación de la experiencia Draft y la inclusión de columnas de auditoría son aspectos fundamentales para asegurar una gestión eficiente y segura de los datos. Este proceso destaca la integración de herramientas y tecnologías de SAP para ofrecer soluciones robustas y flexibles en la nube.

Creación de la Tabla de Persistencia Activa:

El proceso comienza con la creación de una tabla de base de datos para gestionar los datos de la aplicación. Esta tabla servirá tanto para la interacción como para el respaldo de los datos. Debe tener las siguientes características:

- Incluir el campo de cliente o mandante.
- Incluir al menos un campo UUID (Universal Unique Identifier), utilizando los elementos de datos estándar del formato UUID que empiezan con **SYSUUID_**.... La generación y asignación de este valor se deja en manos del framework durante el proceso de creación y la llamada al servicio.



- Agregar los campos requeridos en la tabla, utilizando elementos de datos existentes o creando nuevos en el servidor.
- Añadir columnas de auditoría para registrar quién creó el registro, cuándo fue creado, quién realizó la última modificación y cuándo se realizó dicho cambio. Además de agregar una columna extra de fecha para el concepto de la concurrencia o el ETA de las múltiples peticiones que pudieran llegar de las sentencias Odata. Donde se utilizan los elementos de datos estándar que comienzan con **abp_…** abp_creation_user, abp_creation_tstmpl, abp_locinst_lastchange_user, abp_locinst_lastchange_tstmpl y abp_lastchange_tstmpl.

Debido a que en los proyectos de SAP ABAP Cloud, no es posible definir las tablas en modo gráfico a diferencia del SAP GUI. Es fundamental conocer las anotaciones necesarias para la creación y configuración de las tablas.

Anotaciones	
@EndUserText.label	Sirve para asignar una descripción a la tabla
@AbapCatalog.enhancementCategory	Hace referencia a la categoría de ampliación de la tabla y se pueden elegir las siguientes: ● NOT_EXTENSIBLE ● EXTENSIBLE_CHARACTER ● EXTENSIBLE_CHARACTER_NUMERIC ● EXTENSIBLE_ANY
@AbapCatalog.tableCategory	Sirve para indicar qué tipo de categoría tendrá la tabla, se pueden seleccionar las siguientes: ● TRANSPARENT (Tabla transparente). ● GLOBAL_TEMPORARY (Sólo existen mientras la LUW de la base de datos exista).
@AbapCatalog.deliveryClass	Indica la clase de almacenamiento que tendrá la tabla, se pueden seleccionar las siguientes: ● A ● C

	• L • E • G • S
@AbapCatalog.dataMaintenance	Es útil para indicar si la tabla puede recibir actualizaciones, se pueden elegir entre las siguientes opciones: • DISPLAY • ALLOWED • NOT_ALLOWED • RESTRICTED
@Semantics.amount.currencyCode	Anotación semántica que indica que un campo de la estructura aloja valores de tipo moneda. Con dos puntos se hace referencia al campo que esté asignado con un elemento de datos de tipo moneda.
@Semantics.quantity.unitOfMeasure	Anotación semántica que indica que un campo de la estructura albergará valores de tipo cantidad. Con dos puntos se hace referencia al campo que esté asignado con un elemento de datos de tipo unidad de medida.

Ejemplo:

```
@EndUserText.label : 'Database Table '
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table persitence_table_name {
 key client           : abap.clnt not null;
 key component_uuid   : sysuuid_x16 not null;
 component            : data_element_name;
...
 // audit components
 local_created_by     : abp_creation_user;
 local_created_at     : abp_creation_tstmpl;
 local_last_changed_by : abp_locinst_lastchange_user;
 local_last_changed_at : abp_locinst_lastchange_tstmpl;
 last_changed_at      : abp_lastchange_tstmpl;
```

```
}
```

1.2. Persistencia Draft

Las tablas Draft permiten respaldar de manera temporal los documentos o registros hasta que el usuario complete la información necesaria para confirmarlos y guardarlos permanentemente en la tabla de persistencia activa. Este enfoque mejora la experiencia del usuario al evitar la pérdida de datos durante la edición y permite una validación más flexible y precisa. Las tablas Draft son esenciales cuando se desea ofrecer a los usuarios la capacidad de guardar de manera temporal su trabajo mientras completan formularios o registros. Esto es útil para situaciones donde el usuario necesita consultar información adicional antes de finalizar el registro.

Implementación Técnica:

- Para realizar la implementación técnica de estas columnas se deben realizar los pasos a continuación:
- Duplicar la tabla de persistencia activa, agregando el sufijo "D" para indicar que es una tabla Draft.
- Modificar la nomenclatura de los nombres de los campos o componentes de la tabla de base de datos, eliminando los guiones bajos que separan las palabras. Esta acción se realiza porque en el protocolo OData se utiliza una convención de nomenclatura diferente a la de ABAP Tradicional, usando el estilo de **UpperCamelCase**. Esto significa que la primera letra de cada palabra es mayúscula para separar las palabras. En la definición de las tablas de base de datos, los campos no distinguen entre mayúsculas y minúsculas. Al activar los cambios, todos los componentes se colocarán en minúsculas, con todas las palabras en minúsculas en su nombre. Aun así, se eliminan los guiones bajos para intentar seguir con la nomenclatura OData.
- Las tablas Draft requieren campos adicionales para gestionar la temporalidad y la auditoría de los registros. Estos campos se habilitan utilizando un **include,** a un componente con el nombre **“%admin”** para incluir la estructura estándar **sych_bdl_draft_admin_inc**.

Ejemplo:

"%admin" : include sych_bdl_draft_admin_inc;

Que contiene columnas que permiten al framework determinar qué usuario creó el registro, cuándo fue creado, y qué tipo de operación está guardada temporalmente en la tabla Draft.

```
@EndUserText.label : 'Standard-Include für Draft-Verwaltung (BDL Syntax Check)'
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
define structure sych_bdl_draft_admin_inc {

  draftentitycreationdatetime    : sych_bdl_draft_created_at;
  draftentitylastchangedatetime  : sych_bdl_draft_last_changed_at;
  draftadministrativedatauuid    : sych_bdl_draft_admin_uuid;
  draftentityoperationcode       : sych_bdl_draft_operation_code;
  hasactiveentity                : sych_bdl_draft_hasactive;
  draftfieldchanges              : sych_bdl_draft_field_changes;

}
```

Ejemplo:

```
@EndUserText.label : 'Database Table Draft '
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table draft_table_name_d {
 key client          : abap.clnt not null;
 key componentsuuid      : sysuuid_x16 not null;
 components          : data_element_name;
 // audit components
 localcreatedby      : abp_creation_user;
 localcreatedat      : abp_creation_tstmpl;
 locallastchangedby : abp_locinst_lastchange_user;
 locallastchangedat : abp_locinst_lastchange_tstmpl;
 lastchangedat       : abp_lastchange_tstmpl;
"%admin"          : include sych_bdl_draft_admin_inc;

}
```

1.3. Inserción de datos

La inserción de datos en SAP RAP (Restful ABAP Programming) mediante una clase de programación orientada a objetos permite gestionar eficientemente los datos en la tabla de persistencia. Este proceso incluye la generación de UUIDs, la asignación precisa de datos a las columnas, y la inclusión de campos de auditoría. La limpieza previa de datos asegura que no haya conflictos durante la inserción. Con este enfoque, se garantiza que los datos estén listos para ser utilizados en la aplicación RAP, mejorando la integridad y disponibilidad de la información.

Implementar una inserción de datos

Para esta operación es necesario la creación de una clase ABAP que permita insertar registros en la tabla de persistencia, asegurando que los datos estén disponibles para la aplicación.

Entre aspectos más importantes del proceso tenemos:

- **Limpieza de Datos Preexistentes:** Se utilizan instrucciones **DELETE** para eliminar cualquier dato preexistente en las tablas de persistencia activa y Draft. Esto asegura que no se produzcan errores por claves duplicadas en ejecuciones sucesivas. Las instrucciones **DELETE,** eliminan todos los datos de las tablas, aunque no deben ser utilizadas en proyectos empresariales sin precaución.
- **Inserción de Datos en la Tabla de Persistencia Activa:** La inserción de datos se realiza en la tabla de persistencia activa, utilizando datos de otra tabla de base de datos o a través de agregarlos manualmente en el código fuente de la clase.
- **Alias en columnas requeridas:** Cuando se utilice otra tabla de base de datos a través de una subconsulta (subquery) para consultar e insertar registros en la tabla de persistencia, es necesario agregar alias a los campos que se desean transferir desde la tabla fuente. Esto asegura que los nombres de los campos coincidan con los de la tabla de destino.

- **Generación de UUIDs:** Para los campos UUID, se genera un identificador único utilizando la función **uuid( ).** Que se asigna como un alias en los campos de dicho tipo.

Ejemplo:

```
CLASS class_name DEFINITION
 PUBLIC
 FINAL
 CREATE PUBLIC .
 PUBLIC SECTION.
   INTERFACES if_oo_adt_classrun .
 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.
CLASS class_name IMPLEMENTATION.
 METHOD if_oo_adt_classrun~main.
*  Clean up records in persistence and draft tables
   DELETE FROM persistence_table_name.
   DELETE FROM draft_table_name.
 *  Inserting records into the persistence table
   INSERT persistence_table_name FROM (
   SELECT FROM /dmo/travel
   FIELDS  uuid( ) AS component_uuid,
         comp_persis_name AS  comp_draft_name
   WHERE conditions ).
   IF sy-subrc EQ 0.
    out->write( |Travel.... { sy-dbcnt } rows inserted| ).
   ENDIF.
 ENDMETHOD.
ENDCLASS.
```