



Teoría

EML – Acciones y Características

ABAP RESTful – Arquitectura Cloud





Contenido

1. EML – Acciones y Características	3
1.1. Entidades – Lectura	3
1.2. Entidades – Modificación	7
1.3. Acción con Result	8
1.4. Acción con Parámetros	10
1.5. Características – Feature Instance	11



1. EML – Acciones y Características

1.1. Entidades – Lectura

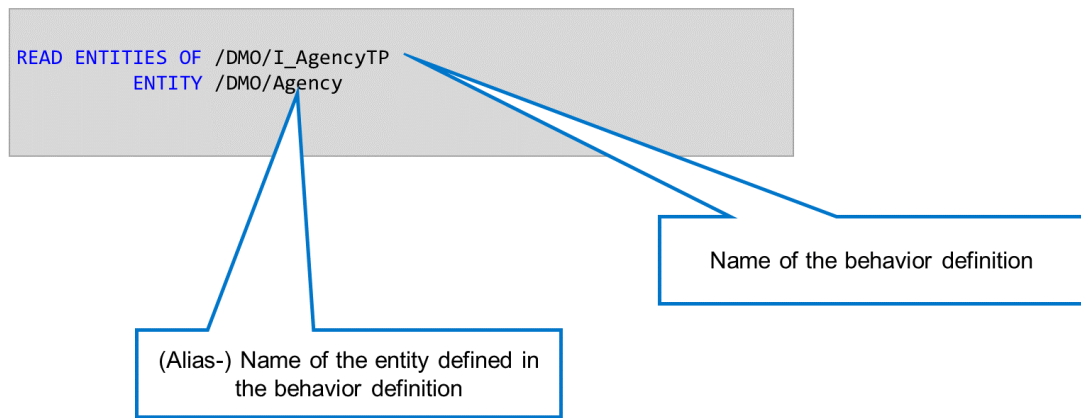
El Entity Manipulation Language (EML), permite manejar las operaciones de negocio de manera eficiente y segura, manteniendo la coherencia transaccional. A través de la implementación en la clase Behavior Pool, los desarrolladores pueden definir y gestionar la lógica de negocio sin necesidad de utilizar comandos SQL directos, asegurando que las operaciones se realicen dentro de un contexto transaccional. Por lo que el framework RAP se encarga de gestionar las transacciones, garantizando la coherencia y consistencia de los datos. Lo que significa que no se deben utilizar comandos COMMIT o ROLLBACK explícitamente.

Operaciones de Lectura:

La lógica necesaria para las operaciones de negocio se realiza en la clase Behavior Pool. Donde se navega a la sección **local types** para implementar los métodos correspondientes.



No se recomienda utilizar comandos SQL directos como SELECT para acceder a los datos. En su lugar las operaciones de lectura se realizan especificando el nombre del **Behavior Definition**, utilizando la sentencia **read entities of** para acceder a las entidades de negocio, por medio de una entidad raíz. Aunque el modelo de datos pueda tener múltiples niveles, la entidad raíz actúa como el punto de entrada principal.



Sintaxis:

```

READ ENTITIES OF root_entity_name IN LOCAL MODE
ENTITY alias_root_entity_name
ALL FIELDS
WITH CORRESPONDING #( keys )
RESULT DATA(lt_root_entity)
FAILED failed.

```

Explicación Detallada:

Para la construcción de la consulta a la entidad raíz mediante la instrucción **read entities of**, se utiliza una serie de instrucciones y sentencias que conforman la lógica para la extracción de datos. A continuación, se explicarán de manera detallada:

- **IN LOCAL MODE:** Se utiliza para leer los datos de las entidades que ya están instanciadas en la memoria en lugar de realizar una lectura directa desde la base de datos. Esto es particularmente útil dentro del contexto transaccional de RAP, donde se pueden tener datos en memoria que aún no han sido confirmados en la base de datos (por ejemplo, datos en modo borrador o cambios pendientes).
- **ENTITY:** Hace referencia al alias de la entidad raíz utilizada en la consulta. Aunque si no se ha definido un alias para la entidad raíz se coloca el mismo nombre de la entidad.
- **ALL FIELDS:** Indica que se van a leer todos los campos de la entidad.



- **WITH CORRESPONDING #(keys)**: Mapea los valores del parámetro **keys** a la estructura esperada por la entidad.
- **RESULT**: Se utiliza para almacenar los resultados de la lectura en una tabla interna la cual es de un tipo especial que se utiliza para la actualización de datos. Dichas tablas de tipo update se declaran de la siguiente manera:

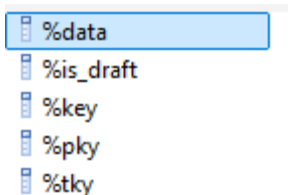
Sintaxis:

DATA lt_table **TYPE TABLE FOR UPDATE** root_entity_name.

También se puede utilizar declaraciones en línea para que el sistema cree la tabla interna con el tipo necesario para la consulta de las entidades.

El resultado de la consulta de la entidad en la tabla interna del tipo update incluirá, además de los componentes definidos en la entidad raíz, los siguientes campos adicionales para gestionar las entidades:

- **%is_draft**: Indica si la instancia de la entidad está en modo borrador (draft). Determinando si los datos son preliminares y pueden ser modificados sin confirmación.
- **%key**: Combinación de las claves primarias que identifican de manera única una instancia de la entidad. Identificando de manera única la entidad para operaciones específicas.
- **%pky**: Clave primaria persistente de la instancia de la entidad. Identifica el registro en la base de datos de manera única.
- **%tky**: Clave técnica de la instancia de la entidad. Gestiona las operaciones transaccionales dentro de RAP.
- **%data**: Este campo representa los datos completos de la entidad en su forma más detallada. Es una estructura que incluye todos los campos y componentes definidos en la entidad raíz, así como cualquier información adicional que sea pertinente para la operación en curso.



- **FAILED failed:** Esta instrucción almacena en la tabla interna **failed**, los detalles sobre los errores específicos que ocurrieron, lo cual es útil para realizar un seguimiento de los problemas y manejar los errores de manera adecuada en el flujo de trabajo.

Parámetro keys:

El parámetro **keys** es una tabla interna que contiene las claves técnicas necesarias para identificar de manera única las instancias de las entidades de negocio que se van a manipular. Esta tabla interna se encuentra definida por defecto como parámetro **importing** de los métodos que incluyen la lógica de las operaciones definidas en la entidad raíz.

La tabla interna **keys** juega un papel fundamental en la identificación y selección de las instancias específicas de las entidades de negocio que se van a manipular. Permite que las operaciones se realicen de manera precisa y eficiente, asegurando que solo los registros correctos sean leídos, modificados, creados o eliminados.

Dentro de la tabla interna **keys**, al igual que las tablas del tipo **update**, se encuentran varios campos importantes utilizados para gestionar las entidades, tales como **%is_draft**, **%key**, **%pky** y **%tky**. Además, incluye el campo:

- **%cid_ref:** Referencia a un conjunto de instancias asociadas a una acción o una entidad relacionada. Manteniendo la relación entre la entidad principal y las entidades dependientes.

1.2. Entidades – Modificación

El uso de EML en RAP permite manejar las operaciones de modificación de manera eficiente y segura, manteniendo la coherencia transaccional. A través de la implementación en la clase Behavior Pool, los desarrolladores pueden definir y gestionar la lógica



de negocio sin necesidad de utilizar comandos SQL directos, asegurando que todas las operaciones se realicen en el contexto adecuado.

Operaciones de Modificación:

Similar a las operaciones de lectura, las modificaciones se realizan en la implementación de los métodos de la sección **local types**. Aunque en esta ocasión este tipo de operación de modificación se realiza utilizando **modify entities of** para acceder y manipular las entidades de negocio.

Sintaxis:

```
READ ENTITIES OF root_entity_name IN LOCAL MODE  
ENTITY alias_root_entity_name  
UPDATE FIELDS ( field field2 ... )  
  WITH internal_update_table.
```

```
MODIFY ENTITIES OF root_entity_name IN LOCAL MODE  
ENTITY alias_root_entity_name  
UPDATE FIELDS ( field field2 ... )  
  WITH VALUE #( for ls_key in keys ( %tky = ls_key-%tky  
                                     field = 'Value' ) ).
```

Donde la sentencia **UPDATE FIELDS**, se utiliza para especificar entre paréntesis y separados por un espacio los campos que serán modificados.

Con la sentencia **WITH VALUE**, se especifica el campo y el valor que se desea modificar en la entidad. Por medio de una instrucción **FOR**, se puede recorrer los elementos dentro de la entidad, utilizando el campo **%tky**, con la clave técnica de la instancia y luego los campos a modificar. Además que por medio de dicha instrucción se pueden agregar condiciones para la consulta.

Para evitar el uso de valores codificados directamente en el código (hardcoding), es una buena práctica definir constantes que representen los valores posibles de ciertos campos.



1.3. Acción con Result

La implementación de acciones que devuelven resultados es crucial para informar al framework sobre los cambios aplicados a las entidades de negocio. Las acciones permiten ejecutar operaciones específicas que modifican el estado de una entidad y deben devolver un resultado al framework para asegurar que los cambios se reflejen adecuadamente.

Las acciones se definen en el Behavior Definition vinculada a una entidad raíz y se implementan en la clase Behavior. Las acciones se definen en el Behavior Definition vinculada a una entidad raíz y se implementan en la clase Behavior Pool. Recordando que se realiza por medio de la instrucción:

action actionName **result [1] \$self;**

La implementación de dicha acción se realiza en un método específico dentro de la clase del behavior pool en la sección **Local Types**. La identificación de los métodos para las acciones utilizan la siguiente estructura en su definición:

Sintaxis:

```
METHODS actionName FOR MODIFY
  IMPORTING keys FOR ACTION root_entity_name~actionName
  RESULT result.
```

Y por medio del EML, se leen y actualizan los campos de los registros de la entidad raíz. Esto se realiza a través de la devolución de la tabla interna **result**. La cual es fundamental para devolver el resultado de las acciones realizadas sobre las entidades de negocio al framework. Esta tabla contiene la información necesaria para que el framework RAP pueda actualizar y reflejar correctamente los cambios en las entidades. A continuación, te explico qué hace y para qué sirven los campos más comunes de dicha tabla.

Dentro de la tabla interna **result**, hay varios campos importantes para devolver el resultado de las acciones realizadas sobre las entidades de negocio:



Dentro de la tabla interna **result**, al igual que la tabla interna **keys**, se encuentran varios campos importantes utilizados para gestionar las entidades, tales como **%is_draft**, **%key**, **%pky**, **cid_ref** y **%tky**. Además, incluye el campo:

- **%param**: Contiene los detalles actualizados de la entidad después de la acción. Proporciona al framework la información completa de la entidad, incluyendo los nuevos valores de sus campos tras la modificación, creación o eliminación.

Estructura de la Devolución:

Es fundamental devolver los resultados de las acciones al framework utilizando la variable **result**. Después de realizar las modificaciones pertinentes, se consulta la información actualizada de la entidad en una tabla interna. Esta tabla se utilizará para transferir los valores a la tabla **result** mediante una instrucción **FOR**, usando los campos **%tky** y **%param** para especificar la clave técnica del registro de la entidad y el valor completo del registro actualizado, respectivamente. Esto permitirá que el framework actúe sobre la información actualizada en la interfaz de usuario.

Sintaxis:

```
result = value #( for ls_root_entity in lt_root_entity (
%tky = ls_root_entity-%tky
%param = ls_root_entity ) ).
```

1.4. Acción con Parámetros

La implementación de acciones con parámetros permite a los usuarios finales proporcionar entradas específicas que afectan el comportamiento de la acción. Estas entradas se utilizan para modificar los datos de las entidades de negocio de acuerdo con los valores proporcionados por los usuarios.



Discount

Discount %:

Discount Cancel

Los parámetros especificados por el usuario están definidos en la entidad abstracta especificada en el Behavior Definition vinculada a una entidad raíz. Recordando que se realiza por medio de la instrucción:

action (features : instance, authorization : update) actionName
parameter absytact_entity_name **result [1] \$self;**

Al agregar lógica dentro del método de las acciones con parámetros, la tabla interna **keys** se amplía para incluir el campo **%params**, y su tipo es el mismo que de la entidad abstracta. Este campo almacena los parámetros especificados por el usuario, definidos en la entidad abstracta. Sin embargo, es necesario utilizar el campo **%tky** para identificar el registro exacto en la tabla mediante la clave técnica. Para obtener esta clave, se debe realizar una lectura de la entidad raíz dentro de la implementación del método de la acción con el parámetro en la clase Behavior Pool.

Para luego realizar una iteración de los registros obtenidos para obtener la clave que se utilizará para devolver el parámetro especificado por el usuario en los parámetros de entrada. Estas iteraciones se realizan mediante un loop y asignando cada registro a un field symbol.

Ejemplo

```
READ ENTITIES OF root_entity_name IN LOCAL MODE
ENTITY alias_root_entity_name
ALL FIELDS
WITH CORRESPONDING #( keys )
RESULT DATA(lt_root_entity).
```



```
loop at lt_root_entity assigning field_symbol_name.  
    DATA(lv_param) = keys[ KEY id  
        %tky = <field_symbol_name>-%tky ]-%param-parameter.  
endloop.
```

Es posible declarar tablas internas del tipo update dentro del método para que puedan ser llenadas en dichas iteraciones por medio del campo clave técnico **%tky**, para luego ser utilizadas directamente en la modificación de los registros de la entidad por medio de la instrucción:

```
DATA: lt_table TYPE TABLE FOR UPDATE root_entity_name.  
APPEND VALUE #( %tky = <field_symbol_name>-%tky  
    Component = 'Value' ) TO lt_table.
```

Por último, es necesario modificar los registros de la entidad raíz y mostrar los registros actualizados en la interfaz de usuario. Esto se logra mediante una lectura posterior a la modificación, para luego transferir dichos valores a la tabla **result**.

1.5. Características – Feature Instance

Las características de comportamiento en RAP (ABAP RESTful Application Programming) permiten modificar visualmente la aplicación en la interfaz de usuario. Estas características pueden habilitar, deshabilitar, poner en modo solo lectura o levantar restricciones sobre los campos y las acciones disponibles en la interfaz, basándose en los valores de ciertas propiedades de la entidad.

Implementación de características:

Las características de comportamiento se pueden aplicar a diferentes objetos, tanto a campos (Field) como a acciones (Action), que posean la característica **features : instance**. Esta propiedad permite consolidar o tener en cuenta cambios de características visuales en



la aplicación y se agrega en la definición dentro del Behavior Definition. La lógica vinculada a los campos de estas características se implementa en la clase Behavior Pool, donde se interpretan y aplican las características definidas mediante el método **getInstanceFeatures** para consolidar los cambios de características en los tipos de datos.

La tabla interna **result**, que se utiliza para devolver los resultados de la acción al framework, se amplía dentro del método **getInstanceFeatures**, para incluir los campos: **%field** y **%action**, que representan los campos y las acciones respectivamente. Estos campos están vinculados con los objetos Field y Action, los cuales tienen la característica de comportamiento **features : instance** en la Behavior Definition. Mediante esta tabla interna, al iterar sobre los registros leídos de la entidad raíz, se pueden configurar las características de comportamiento de los campos y las acciones. Esto se logra estableciendo en los campos **%field** y **%action**, los valores de las constantes del comportamiento a través de la interfaz **if_abap_behv**.

Configuración de los campos de comportamiento:

El campo **%field**, permite especificar los componentes de la entidad los cuales tengan la característica de comportamiento. Para luego agregarles una configuración de acuerdo con una condición. Los posibles valores que se pueden utilizar son los siguientes:

- **read_only**: Pone el campo en modo solo lectura, evitando que se realicen modificaciones.
- **unrestricted**: Levanta cualquier restricción previamente aplicada al campo, permitiendo su edición.
- **mandatory**: Define el campo como obligatorio, requiriendo que se complete antes de proceder.
- **all**: Aplica todas las características anteriores según el contexto y la lógica definida.

De forma similar el campo **%action**, permite especificar las acciones que tengan la característica de comportamiento. Para luego agregarles una configuración de acuerdo con una condición. Los posibles valores que se pueden utilizar son los siguientes:



- **if_abap_behv=>fc-o-disabled:** Deshabilita la acción, haciendo que el botón correspondiente no sea interactivo.
- **if_abap_behv=>fc-o-enabled:** Habilita la acción, permitiendo que el botón correspondiente sea interactivo.

Recordando que es necesario modificar los registros de la entidad raíz y mostrar los registros actualizados en la interfaz de usuario. Esto se logra mediante una lectura posterior a la modificación, para luego transferir dichos valores a la tabla **result**.

Ejemplo:

```

READ ENTITIES OF root_entity_name IN LOCAL MODE
ENTITY alias_root_entity_name
ALL FIELDS
WITH CORRESPONDING #( keys )
RESULT DATA(lt_root_entity)
FAILED failed.

```

```

result = VALUE #( FOR ls_root_entity IN lt_root_entity (
    %tky =      ls_root_entity-%tky
    %field-entityComponent = COND #(
        WHEN ls_root_entity-entityComponent = 'Value'
        THEN if_abap_behv=>fc-f-read_only
        ELSE if_abap_behv=>fc-f-unrestricted )
    %action-actionName = COND #(
        WHEN ls_root_entity-entityComponent = 'Value'
        THEN if_abap_behv=>fc-o-disabled
        ELSE if_abap_behv=>fc-o-enabled )
    ) ).

```