

## Aplicação Agência de Viagens

O objetivo proposto foi criar uma aplicação para uma agência de viagens como já referido no relatório. Neste anexo é apresentado a construção da aplicação. Neste documento faço uma análise exaustiva de como se constrói, exceto nos casos da construção da parte das reservas, que é igual à da viagem e na parte final no desenvolvimento dos métodos, eu escolhi dois para explicar o raciocínio.

### 1º : Configurar o ambiente no eclipse

O primeiro passo é criar uma conta SAP que, dentro da *service instances*<sub>1</sub> criada do tipo *abap trial*<sub>25</sub> (figura 1), e dentro desta criamos uma nova instância (figura 2). Dentro desta instância criamos uma *key service*<sub>24</sub>. Precisamos desta key, porque é a que vamos usar para configurar manualmente a conexão com a instância criada do ambiente ABAP, bem como com as Ferramentas de Desenvolvimento ABAP no futuro. A seguir, vamos configurar o Eclipse para que fique com a perspectiva de ABAP Development Tools. Após tudo configurado criamos um projeto (“create an ABAP cloud project”) e selecionamos o service key, pois é através do ficheiro json, que foi criado no site SAP no início, que vamos fazer a ligação (figura 3).

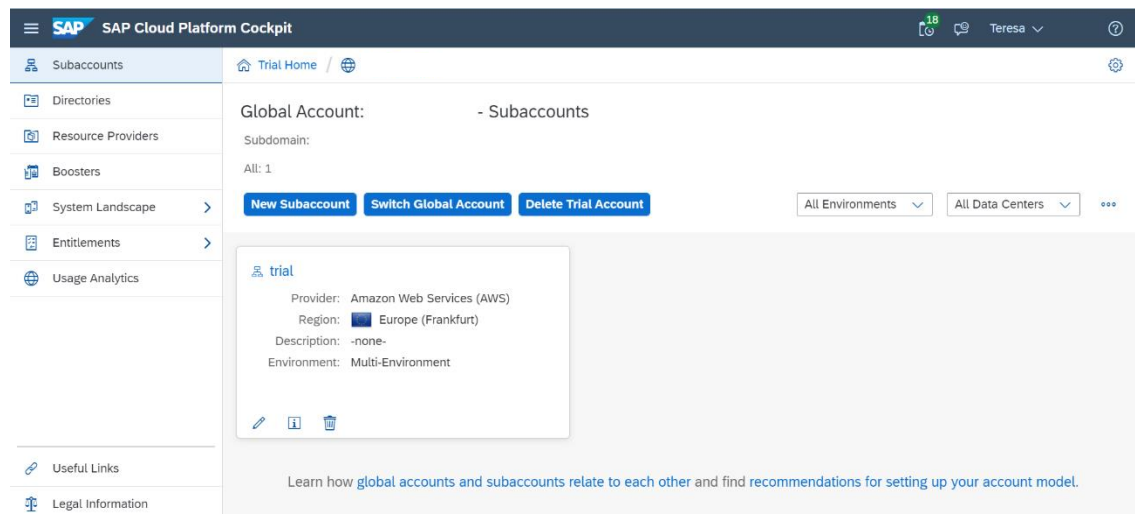


Figura 1: Ambiente SAP com conta trial

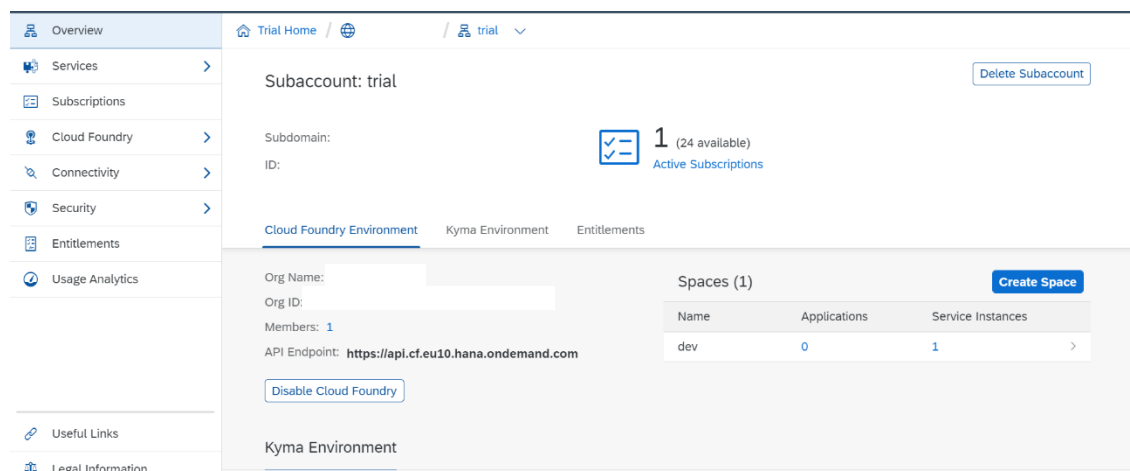


Figura 2: Conta trial com a service instances que se a selecionarmos, vamos poder criar a key.

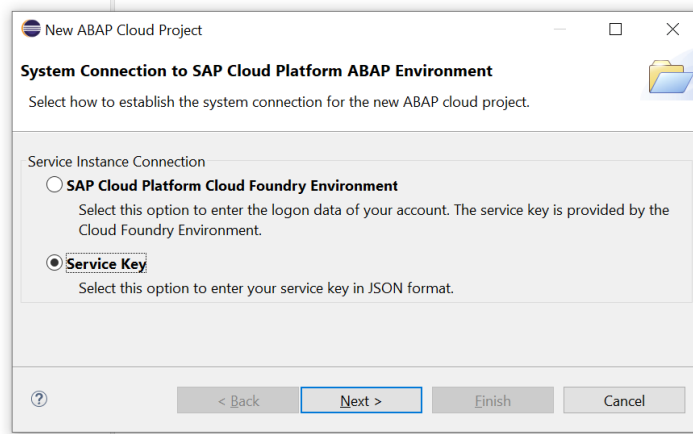


Figura 3: Opções de como é feita a conexão entre o eclipse e o site SAP.

Após fazer essa ligação basta fazer log on (figura 4).

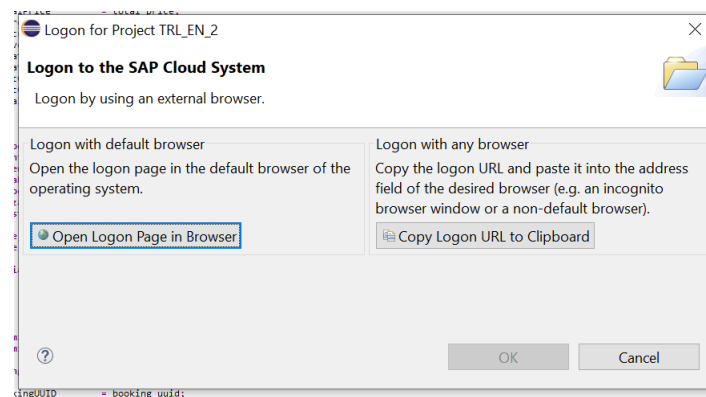
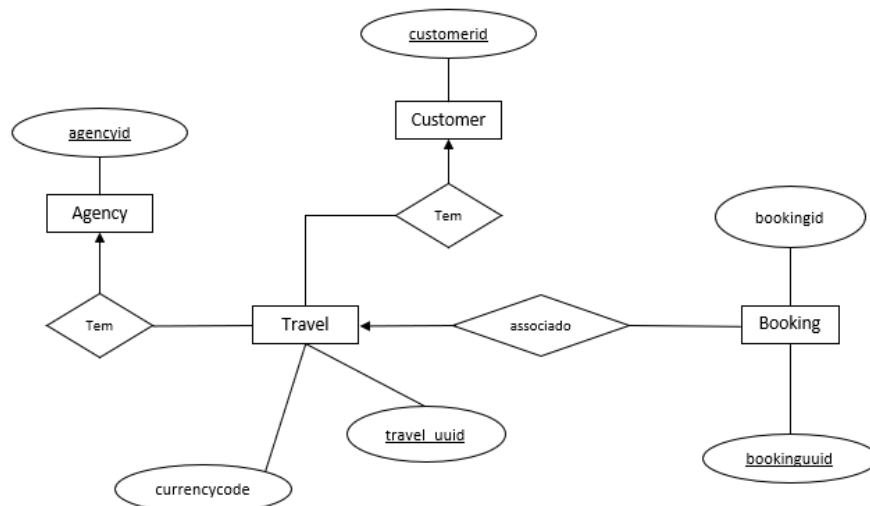


Figura 4: Logon to the SAP Cloud System. Este logon é necessário porque apesar de estarmos a desenvolver no eclipse, fica tudo em cloud.

## 2º : Modelo de dados

Nesta secção do relatório está representado o esboço do modelo de dados completo da aplicação agência de viagens, que permite ver as relações entre as entidades e quais os seus atributos mais relevantes.



### Legenda do esquema do modelo de dados:

Retângulos representam conjuntos de entidades.

Losangos representam conjuntos de relações.

Elipses representam atributos.

Linhas ligam atributos aos conjuntos de entidades e conjuntos de entidades a conjuntos de associações.

Sublinhado representa atributos constituintes da chave primária.

A anotação é baseada no livro da cadeira de base de dados, LCCN 2018060474 | ISBN 9780078022159 (alk. paper) | ISBN 0078022150

No modelo de dados apresentado em cima retiramos que a Travel, o Booking, a Agency e o Customer são entidades. A relação da travel e do booking, a relação da agency e da travel, a relação da customer e da travel é de muitos para um. Estas associações, respetivamente, leem-se que uma viagem pode ter zero ou mais do que um booking associado, um booking tem exatamente uma viagem associada e uma agency pode ter zero ou mais viagens associadas, mas a travel só pode ter uma agência associada. A travel é associada exatamente a um customer, mas um customer pode ter mais do que uma viagem.

A entidade travel tem como atributos a travel\_uuid, currencycode, agencyid, customerid. Os atributos travel\_uuid, agencyid e o customerid são chaves primárias.

Relativamente à entidade reserva tem como atributos o booking\_uuid, que é também a chave primária e o bookingid. As todas entidades tem outros atributos, mas não estão aqui representados.

### 3º: Criar tabela de dados das viagens e das reservas

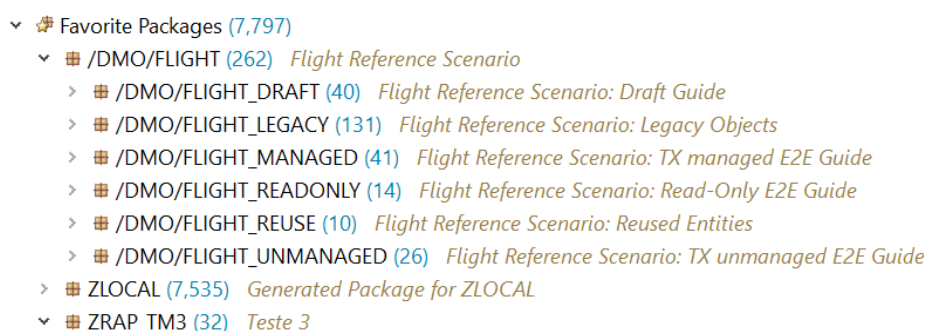
Quando chegamos aqui já temos tudo pronto para poder começar a desenvolver a aplicação. O primeiro passo nesta fase consiste em criar um packet geral “ZLOCAL”, que vamos associar um *transport request*<sub>2</sub> (figura 5), que também terá que ser criado. Este transport request vai estar sempre associado a qualquer funcionalidade que criamos.

Transport Request	Owner	Target	Description	CTS Project
TRLK901087	CB00000...		Teste 3 TM	
TRLK900219	CB00000...		RAP exercise at openSAP tm	

Figura 5: Transport Request

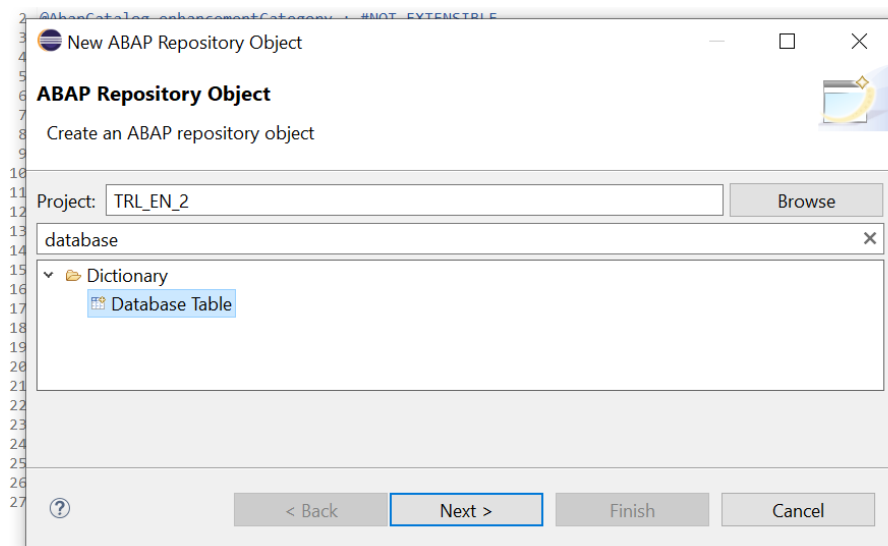
Como referi no relatório, o ponto de partida do desenvolvimento da construção da aplicação (o development flow) é a criação das tabelas viagens e reservas que tem os dados a que se seguem o modelo de dados CDS<sub>4</sub>, a projeção do modelo de dados e o enriquecimento com a semântica da interface do utilizador. Em seguida, criaremos o serviço Odata<sub>10</sub> e visualizaremos a aplicação de viagens SAP Fiori<sub>11</sub>. Finalmente vamos implementar a autorização básica para acesso a dados usando funções CDS.

Antes de começar a criar as tabelas de base de dados é importante perceber que o cenário de referência de voo ABAP pertence ao namespace<sub>12</sub> / DMO /<sub>13</sub>, por isso, é necessário adicionar o pacote principal, / DMO / FLIGHT, aos meus pacotes de favoritos. O cenário de referência contém exemplos de implementações dos diferentes recursos RAP bem como várias implementações legadas, dictionary objects, tabelas de base de dados, domínios e elementos de dados e dados de demonstração (figura 6).



*Figura 6: DMO, cenário de referência as viagens. Dentro do pacote mais geral dos dados das viagens estão outros pacotes que contém os dados das viagens que ficam guardadas como rascunhos, os dados das viagens de uma aplicação managed ou unmanaged.*

Começamos então por criar duas tabelas (Travel e Booking) de base de dados onde vão ser armazenados os dados das viagens e das reservas (figura 7).



*Figura 7: Mostra a criação da estrutura database.*

Os atributos específicos da tabela e do campo são especificados por meio de anotações. Os campos da tabela são especificados entre as chaves, e a keyword "key" é usada para especificar os campos-chave. Os atributos técnicos da tabela são definidos na parte de cima da definição da tabela, antes da keyword "define". Como se pode ver no screenshot, vemos as anotações usadas para especificar a referência da currency key para o amount fields, booking fee, total price, e flight price. A tabela agora consiste nos key fields, client e travel\_uuid, e table fields, como o travel\_id, o agency\_id, o customer\_id, o total\_price e o overall\_status. Para além destes também inclui os dados de administração standard, como o respectivo utilizador ou, por exemplo a created\_at dos dados. A tabela field currency\_code é especificada como o campo de referência para os campos de valor booking\_fee e total\_price usando uma anotação semântica. Se correr a aplicação como está ainda não aparece nada, porque a travel list report está vazia. A solução para resolver esse problema é criar uma *ABAP Class*<sup>26</sup>. A anotação *@AbapCatalog* define as configurações técnicas de entidades CDS no *Dicionário ABAP*<sup>27</sup>, existem vários tipos de anotações relacionados (figura 8).

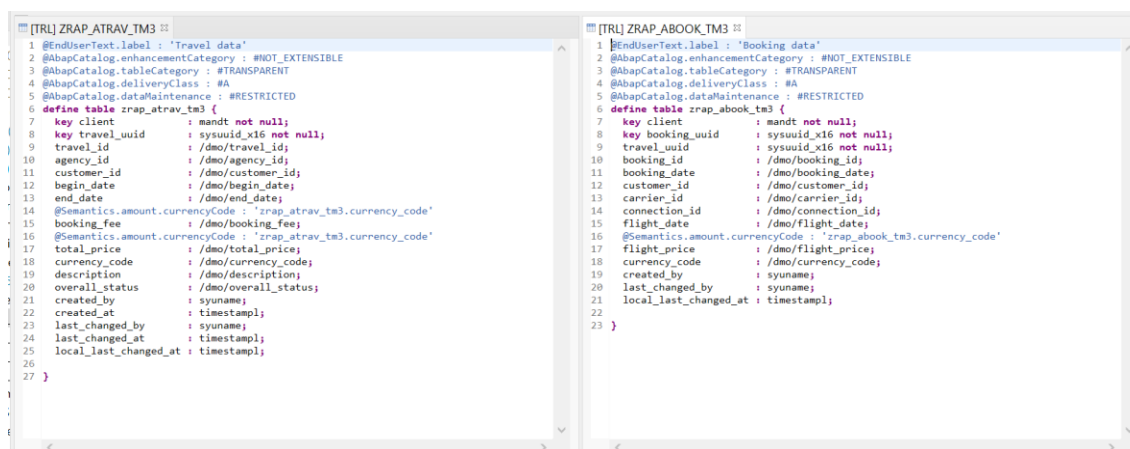


Figura 8: Mostra a configuração das tabelas de base de dados da viagem (lado esquerdo) e as reservas (lado direito).

Na criação desta classe (figura 9) vamos adicionar a interface *IF\_OO\_ADT\_RUN*<sup>14</sup>. Relativamente ao código aqui apresentado, primeiro quaisquer entradas existentes em ambas as tabelas são apagadas (linha 16 e 17). Depois os dados são selecionados da tabela / dmo / viagens e inseridos na nova tabela de viagens. A função SQL uuid () é usada para definir o valor do key field travel\_uuid. O trabalho de commit statement é executado para manter os dados. Neste caso a seleção de dados foi limitada até 200 registos de viagem.

```

1 CLASS zcl_generate_demo_data_tm3 DEFINITION
2 PUBLIC
3 FINAL
4 CREATE PUBLIC .
5
6 PUBLIC SECTION.
7 INTERFACES if_oo_adt_classrun.
8 PROTECTED SECTION.
9 PRIVATE SECTION.
10 ENDCLASS.
11

```

```

12= CLASS zcl_generate_demo_data_tm3 IMPLEMENTATION.
13= METHOD if_oo_adt_classrun~main.
14
15     " delete existing entries in the database table
16     DELETE FROM zrap_atrav_tm3.
17     DELETE FROM zrap_abook_tm3.
18
19     " insert travel demo data
20     INSERT zrap_atrav_tm3 FROM (
21         SELECT
22             FROM /dmo/travel
23             FIELDS
24                 uuid( )          AS travel_uuid          ,
25                 travel_id        AS travel_id            ,
26                 agency_id        AS agency_id            ,
27                 customer_id      AS customer_id          ,
28                 begin_date       AS begin_date           ,
29                 end_date         AS end_date              ,
30                 booking_fee      AS booking_fee           ,
31                 total_price      AS total_price           ,
32                 currency_code    AS currency_code         ,
33                 description      AS description           ,
34                 CASE status
35                     WHEN 'B' THEN 'A' " accepted
36                     WHEN 'X' THEN 'X' " cancelled
37                     ELSE 'O'         " open
38                 END              AS overall_status        ,
39                 createdby        AS created_by            ,
40                 createdat        AS created_at            ,
41                 lastchangedby    AS last_changed_by       ,
42                 lastchangedat    AS last_changed_at       ,
43                 lastchangedat    AS local_last_changed_at ,
44                 ORDER BY travel_id UP TO 200 ROWS
45     ).
46     COMMIT WORK.
47
48     " insert booking demo data
49     INSERT zrap_abook_tm3 FROM (
50         SELECT
51             FROM /dmo/booking AS booking
52             JOIN zrap_atrav_tm3 AS z
53             ON booking~travel_id = z~travel_id
54         FIELDS
55             uuid( )          AS booking_uuid          ,
56             z~travel_uuid    AS travel_uuid           ,
57             booking~booking_id AS booking_id           ,
58             booking~booking_date AS booking_date       ,
59             booking~customer_id AS customer_id         ,
60             booking~carrier_id AS carrier_id           ,
61             booking~connection_id AS connection_id     ,
62             booking~flight_date AS flight_date         ,
63             booking~flight_price AS flight_price        ,
64             booking~currency_code AS currency_code     ,
65             z~created_by      AS created_by            ,
66             z~last_changed_by AS last_changed_by       ,
67             z~last_changed_at AS local_last_changed_by ,
68     ).
69     COMMIT WORK.
70
71     out->write( 'Travel and booking demo data inserted.' ).
72     ENDMETHOD.
73
74 ENDClass.

```

Figura 9: ABAP Class que vai permitir correr e ler as bases de dados criadas.

#### 4º: Definir o modelo de dados Core data services

Depois de já termos as bases de dados criadas e a funcionar, começamos por definir o modelo de dados CDS clássico e, depois, melhoramo-lo para definir a estrutura do *business object*<sub>5</sub> (BO). Assim, começamos com a criação da *view*<sub>15</sub> da viagem: clica-se na tabela da viagem e selecciona-se *New Data Definition*<sub>16</sub>. A tabela das viagens é usada como fonte de dados e os campos da tabela são inseridos automaticamente na lista das projecções entre as chaves. A seguir, definimos os *alias*<sub>18</sub> para a fonte de dados e visualizamos as colunas usando a keyword "as" e melhoramos o modelo de dados CDS com associações para definir a sua relação com outras entidades.

As associações às entidades Booking, Agency, Customer e Currency são definidas e expostas na lista de *projeções*<sup>17</sup>. Uma associação é definida com uma cardinalidade, uma entidade CDS de destino, um *alias*<sup>18</sup> opcional e uma condição on. Na lista de projeções, o elemento de visualização *overall\_status* é renomeado para *TravelStatus*.

Depois, podemos adicionar *semantics annotations*<sup>19</sup> ao currency e administrative fields para garantir um processamento de dados uniforme do lado do consumidor. Podemos também especificar o *currencyCode* como o campo de referência dos campos de currency BookingFee e TotalPrice usando uma semantics annotations. A anotação dos administrative fields CreatedBy, CreatedAt, LastChangedBy, LastChangedAt e LocalLastChangedAt são uma etapa de preparação para uma próxima etapa no futuro, onde o *transaccional behaviour* da aplicação *Travel List Report* será ativado. Para estes administrative fields são necessárias anotações para permitir a atualização automática dos campos de administração em cada operação. Aplica-se o mesmo raciocínio na criação do *booking view*. Aqui no screenshot estão os dois códigos relativamente às viagens e às reservas (figura 10).

Para terminar esta parte vamos definir a estrutura do nosso *business objects*<sup>5</sup>. Esta etapa não é obrigatória na criação da aplicação do tipo *read-only*, mas fazemo-la já para preparar o ambiente, o transactional enablement da nossa aplicação de viagem, às necessidades futuras. Assim, na parte da view da viagem, especifico a entidade de viagem como o nó root do BO composition usando a keyword "root" (linha 3 da travel). Na linha 6 da travel deve-se especificar a view da reserva como "filho" do BO usando a associação especial, composition. Relativamente ao booking temos que mudar a view da reserva e especificar a entidade viagem como nó "pai" usando a associação especial, association to parent (linha 6 do booking). (figura 10).

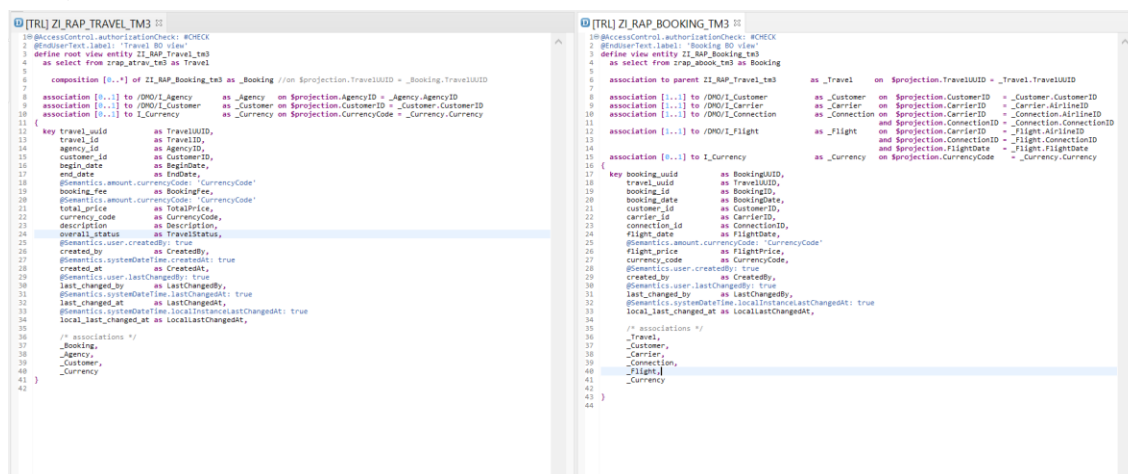


Figura 10: Data definition.

O próximo passo é projetar a parte do CDS que nos é realmente importante para a nossa aplicação. O objetivo é termos o modelo de dados CDS construído a partir de duas projeções do CDS views: uma view de projeção da viagem e uma view de projeção da reserva.

No screenshot (figura 11), a linha 3, a anotação *@Search*, representa a ativação da projeção para a pesquisa e especifica o element view para ser considerado numa pesquisa. A linha 4 está a permitir que haja facilidade nas modificações de ambas as *projections views*<sup>17</sup> com as anotações de UI num documento separado: este documento é a extensão dos metadados CDS (em suma MDEs) que os separa das anotações relacionadas com o business objects. Através da anotação *@ObjectModel* vamos adicionar alguns novos campos, como *Agency\_Name*, *customer\_Name* e o *Carrier\_Name*, na lista de projeção e

estabelecer a conjunção de um campo com as suas unidades de texto descritivo independente da lingua. Também definimos o *value help*<sub>20</sub> para os campos Agency ID, Customer ID, and Carrier ID usando a anotação *@Consumption*, e redirecionamos os nós de composition da viagem e da reserva para as entidades correspondentes da camada de projeção do business object. Com as projections views, podemos expôr apenas aqueles elementos que são relevantes para o serviço específico, incluindo a desnormalização do modelo de dados subjacente. O *value help*, a pesquisa e a semântica da UI também podem ser definidos.

A seguir, temos que especificar a view da projeção do BO como a entidade root, e fornecer um alias para a projection view da viagem que já está atribuída. Todas as views, elementos e associações são inseridos automaticamente na lista de projeção.

Nesta fase, vamos adicionar algumas semânticas específicas do serviço ao modelo de dados projetado, porque permite que a projection view seja aprimorada com extensões de metadados separadas e porque permite a pesquisa usando as anotações *@Metadata.allowExtensions* e *@Search.searchable*.

Relativamente à lista de projeções, a anotação *@Search.DefaultSearchElement* ativa a pesquisa das colunas TravelID, AgencyID e CustomerID. A anotação *@ObjectModel.text.element* especifica os elementos adicionados anteriormente AgencyName e CustomerName como descrições para os elementos AgencyID e CustomerID, respectivamente. A anotação *@Consumption.ValueHelpDefinition* define o value help para os elementos de visualização AgencyID, CustomerID e CurrencyCode. Aqui, o nome da entidade CDS que atua como um value help, é o nome do elemento que está vinculado ao elemento local que tem que ser especificado. É possível completar automaticamente ao especificar a entidade do value help de destino.

Vamos também especificar o elemento currency como o campo de referência para os campos de currency BookingFee e TotalPrice, usando o elemento annotation *@Semantics.amount.currency*. Manteremos todas as associações expostas para caso seja necessário usar, mas redirecionaremos o BO “filho” da entidade booking para a visualização de projeção do BO Booking. A anotação *@ObjectModel.text.element* serve para fornecer uma descrição aos elementos CustomerId e CarrierId. Para o CustomerID, o CarrierID, o ConnectionID e o CurrencyCode os *value help* são definidos de formas diferentes. Para o *value help* do ConnectionID temos que adicionar uma condição de ligação extra, que é definida para retornar valores do *value help* que já foram guardados para os elementos de visualização locais CarrierID, FlightDate, FlightPrice e Currency (linha 23 do booking). O campo de visualização CurrencyCode é especificado como o campo de referência para o campo de currency FlightPrice (linha 31 booking).



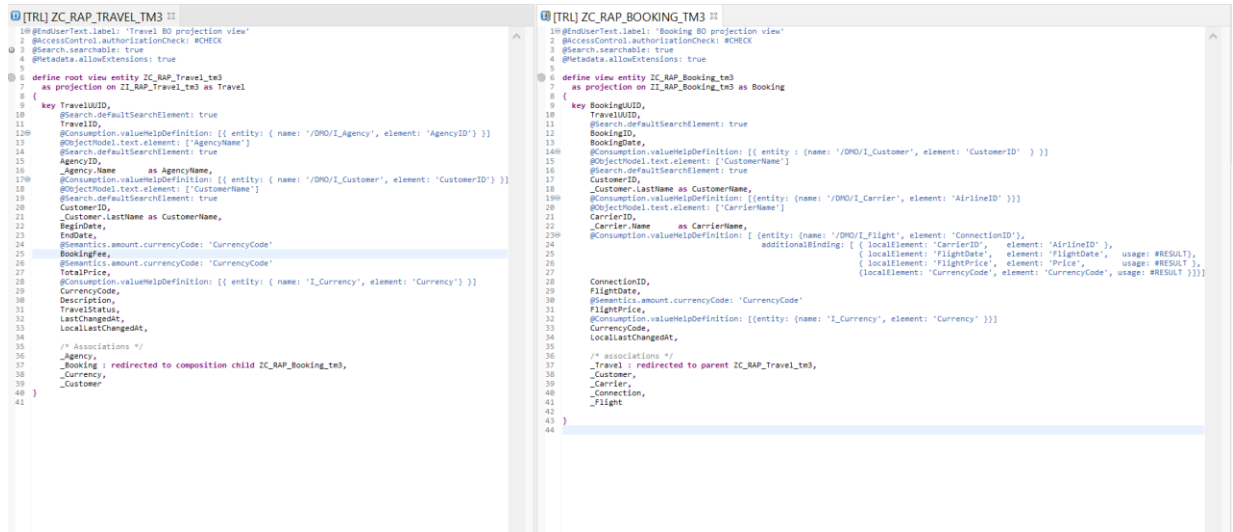


Figura 11: Projection views.

De modo a enriquecer ainda mais a projeção do modelo de dados CDS com a semântica da UI vamos usar extensões de metadados CDS (MDEs), (figura 12). As extensões dos metadados do CDS MDEs permitem que se melhore sem alterações no modelos de dados. Com o CDS, podemos adicionar semânticas para diferentes domínios para uma entidade CDS existente usando anotações, normalmente, anotações de UI. Ao usar CDS MDEs, conseguimos uma separação do modelo de dados com todos os business objects da semântica da UI. Essa separação de interesses permite uma melhor gestão de mudança simplificada sem modificação. Graças à abordagem por causa das camadas de CDS MDEs, podemos definir mais de uma extensão de metadados CDS para uma entidade CDS. Ficamos assim com duas extensões de metadados CDS: um MDE CDS para a view de viagem projeção e outro para a projeção de reserva.

Cada extensão de metadados é atribuída a uma camada, como *CORE#*, *PARTNER#* ou *CUSTOMER#*. A camada determina a prioridade da avaliação. Então com a *view* diversificada e os campos das anotações da UI específicos, podemos ver o nome da entidade CDS que está enriquecida. Nesta altura, temos a anotação *@Metadata.layer*, o nome da projection view do BO após a instrução *ANNOTATE VIEW* e uma entrada fictícia entre as chaves. Devemos especificar o *#CORE* como a camada de metadados. Quando existem várias extensões de metadados definidas para uma determinada entidade CDS, a layer determina a prioridade dos metadados. *#CORE* tem a prioridade mais baixa e *#CUSTOMER* a prioridade mais alta. Em cima devemos definir também algumas informações do cabeçalho, como o nome do tipo e o título. Neste caso, os dados de viagem apresentados serão classificados em ordem decrescente pelo elemento *TravelID* na lista, por isso, nas chaves, estamos a usar *@UI.facet* anotações para definir a pesquisa para a página do objeto e do seu layout. A página de objetos de viagem tem duas facetas: a referência de identificação das entidades de viagens e a referência do item da linha da entidade do booking, com a *composition\_booking* especificada como elemento de destino.

A seguir, devemos especificar uma posição e possivelmente um rótulo para cada elemento usando as respectivas anotações de elemento *@UI*. O *@UI.lineItem* é a anotação usada para especificar as informações de layout de cada elemento mostrado como uma coluna na lista. A anotação *@UI.identification* é usada para especificar as informações de layout de cada elemento mostrado na secção de identificação da página do objeto. E a anotação *@UI.selectionField* é usada para permitir que

um elemento seja selecionado na funcionalidade filtro. Com a anotação `@UI.hidden: true`, evitamos que os elementos sejam exibidos na UI ou nas configurações. Os elementos `TravelID`, `AgencyID` e `CustomerID` estão disponíveis para serem selecionado nos filtro e também mostrados na lista de viagens.

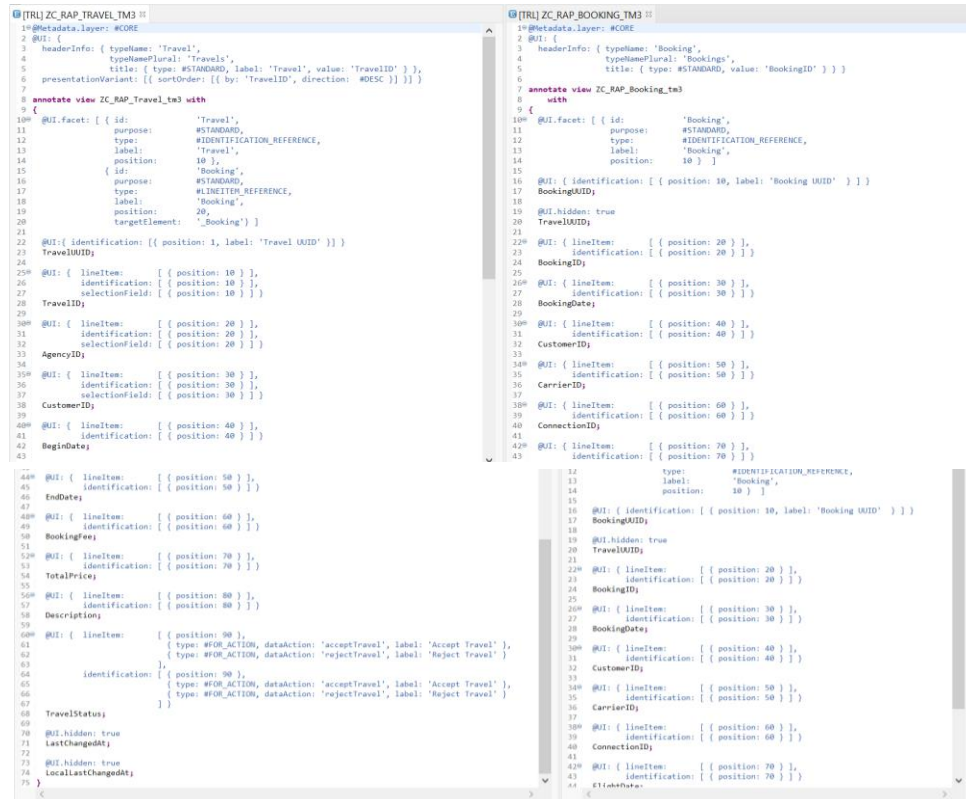


Figura 12: Metadados do CDS.

## 5º: Serviço Odata

O próximo passo é criar o serviço Odata e visualizar a aplicação. Passos a seguir: criar uma *service definition*<sub>8</sub> (figura 13) para especificar o scope do serviço *Odata*, e de seguida criar uma *service binding*<sub>9</sub> para ligar o *service definition* ao protocolo *OData* como um serviço de UI. O scope do serviço é definido expondo o que é relevante: cds views e os metadados.

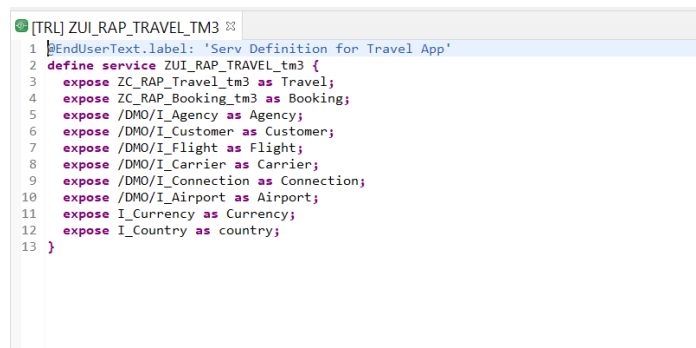


Figura 13: Service Definition.

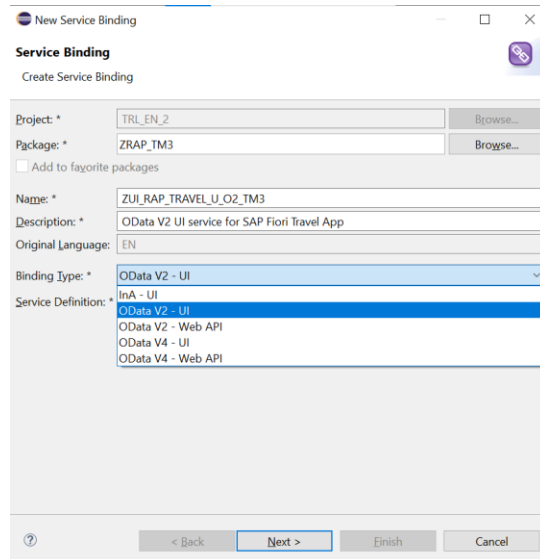


Figura 14: Criação do Service Binding. Atualmente apenas OData V2 é compatível, no entanto, OData V4 está planejado para ser usado.

No *service binding* (figura 15) temos o service URL, os conjuntos de entidades expostas e as associações. Se clicarmos no link do service URL podemos ver os metadados do serviço no browser.

Para podermos ver a aplicação selecionamos a “Travel” e clicamos duas vezes ou clicamos com o botão direito, e aparece a opção abrir com Fiori Elements app.

A aplicação é aberta no browser e podemos ver as colunas de pesquisa: Agency ID e Customer ID que estão de acordo com as anotações de pesquisa definidas na projection view do BO de viagem. Podemos filtrar como por exemplo, pelo ID da agência, podemos usar o *value help*, podemos personalizar a lista adicionando, movendo e removendo colunas através da personalização das configurações. Por exemplo, podemos adicionar o *UUID* e movê-lo para a primeira posição. E podemos também facilmente restaurar o layout padrão.

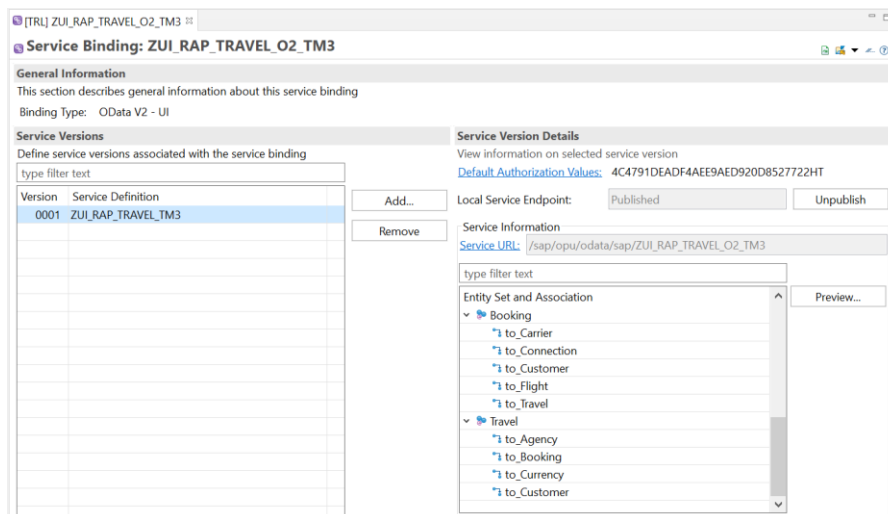


Figura 15: Service Binding.

## 6º : Implementação das autorizações básicas

Para finalizar, vamos implementar uma autorização básica para dados de acesso usando funções CDS para o business objects de viagens. O *ABAP CDS* fornece uma linguagem de controle de dados, DCL, para definir o acesso de autorização aos dados usando uma função CDS. A autorização clássica usada na plataforma ABAP requer verificações explícitas de autoridade codificada nos dados selecionados, que são armazenados em tabelas internas na camada de aplicação. O CDS DCL, em vez disso, oferece uma alternativa declarativa com base em verificações de autorização implícitas que ocorrem durante as tentativas de acesso à entidade CDS. As funções do CDS podem ser definidas usando condições PFCG, condições literais, condições do utilizador e condições de herança. Devem ser definidas explicitamente para cada entidade CDS, quando necessário. Fazem parte do modelo de dados e, portanto, também são enviadas para a base de dados para que apenas os dados autorizados sejam devolvidos ao ler um modelo de dados CDS.

As funções do CDS são definidas uma vez e usadas automaticamente em todo o lado. Ao aceder às entidades CDS, teremos definido um authorization object e duas funções de CDS para a entidade de viagem, uma na camada de modelação de dados e outra na camada de provisionamento de business service para a projeção do modelo de dados. A função do CDS na camada de modelação de dados será definida com condições literais e PFCG, e criaremos um objeto de autorização para o efeito. O *PFCG<sub>28</sub>* é um transaction code cujo objetivo é manter administração para manage roles e authorization data. A segunda função CDS irá herdar a condição da camada subjacente. As regras de acesso consistirão numa condição literal para o elemento de visualização *CurrencyCode* e uma condição PFCG para o elemento de visualização *TravelStatus*. Para a definição do PFCG, criaremos um authorization object, incluindo campo de autorização e elemento de dados.

The screenshot shows the SAP Data Element configuration interface for 'ZOSTAT\_TM3'. The 'Data Type Information' tab is active, showing the following details:

- Category: Domain
- Type Name: DMO/OVERALL\_STATUS
- Data Type: CHAR
- Length: 1

The 'Field Labels' tab is also visible, showing the following field labels and their maximum lengths:

Field Label	Maximum Length
Short: Status	10
Medium: Travel Status	20
Long: Travel Status	40
Heading: Travel Status	55

Figura 16: Criação do Data Element.

No campo Category temos que pôr “Domain” e inserir o nome do tipo no campo Type name. Nos Filed labels como mostra o screenshot, devemos preencher com “Status” e “Travel Status” (figura 17).

**Authorization Field: ZOSTAT\_TM3**

**General**

Data Element: \* ZOSTAT\_TM3

Provide Search Help in Standard Maintenance Dialog

Check Table:

**Used in authorization object**

type filter text

Class	Object	Description
CPAE	ZOSTAT_TM3	Authorization object for travel status

**What's next?**

[Create a new Authorization Object and assign the Authorization Field to it](#)  
[Assign the Authorization Field to an existing Authorization Object](#)

Figura 17: Authorization object.

Neste screenshot estamos a permitir adicionar, criar, alterar, fazer display e apagar (figura 18).

**Authorization Object: ZOSTAT\_TM3**

**General**

Object Class: CPAE Object Class Description: SAP Cloud Platform ABAP Environment Objects

**Authorization Fields**

type filter text

Authorization Field	Description	Activity Field
ZOSTAT_TM3	Travel status	<input type="checkbox"/>
ACTVT	Activity	<input checked="" type="checkbox"/>
<Enter new value>		<input type="checkbox"/>

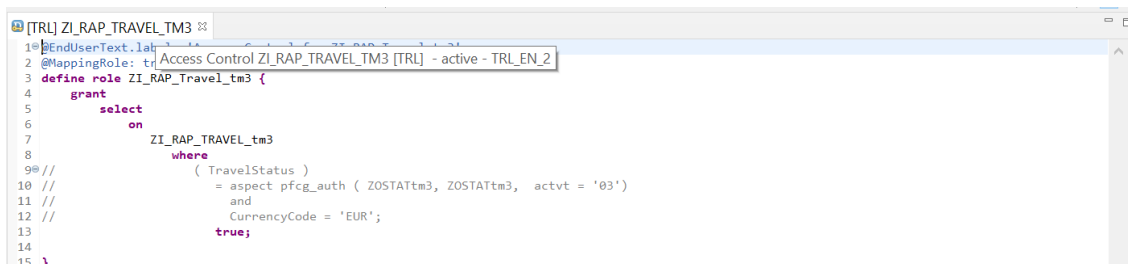
**Permitted Activities**

type filter text

Activity	Description	Access Category
01	Add or Create	Write
02	Change	Write
03	Display	Read
06	Delete	Write
<Enter new value>		

Figura 18: Authorization object, permissões das atividades criar, altrear, fazer display e apagar.

No screenshot da figura 19 estamos a definir o papel do CDS para a view do BO da viagem. Assim, vamos criar um novo controle de acesso. A anotação @MappingRole: true é definida no topo para atribuir a função CDS a todos os utilizadores, independentemente do cliente. O nome do CDS protegido é especificado após o select na declaração e as regras de acesso do utilizador são definidas na cláusula where. Em comentário está definido uma condição no elemento CurrencyCode. Apenas registos com o código de moeda EURO devem ser seleccionados. Está em comentário porque nós queremos ter o acesso total aos dados e para isso podemos adicionar uma condição ou “true” na cláusula where do controle de acesso para o BO de viagem.



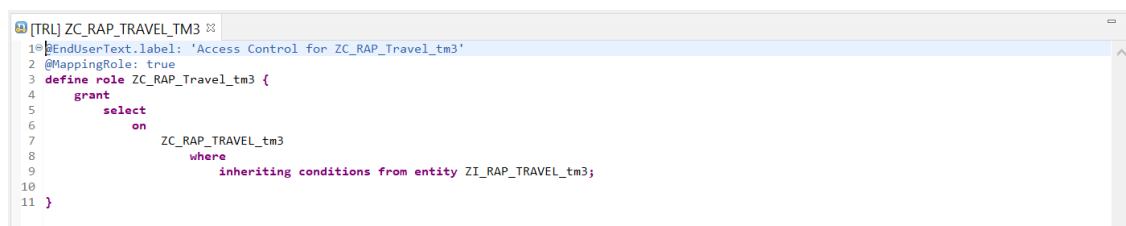
```

1 @MappingRole: true
2 @MappingRole: true
3 define role ZI_RAP_Travel_tm3 {
4   grant
5     select
6       on
7         ZI_RAP_TRAVEL_tm3
8         where
9           // ( TravelStatus )
10          // = aspect pfcg_auth ( ZOSTATtm3, ZOSTATtm3, actvt = '03' )
11          // and
12          // CurrencyCode = 'EUR';
13          true;
14 }
15

```

Figura 19: Definição do papel do CDS para a view do BO da viagem.

Conforme já explicado, uma função CDS deve ser definida explicitamente para cada entidade CDS. Não há herança implícita de regras de acesso. Portanto, vamos agora definir as regras de acesso para a visão de projeção do BO de viagem (figura 20). A anotação @MappingRole: true é definida na parte superior. O nome do CDS é especificado após a instrução grant select on, e, na cláusula da condição where, podemos definir a entidade CDS da qual as condições devem ser herdadas.



```

1 @MappingRole: true
2 @MappingRole: true
3 define role ZC_RAP_Travel_tm3 {
4   grant
5     select
6       on
7         ZC_RAP_TRAVEL_tm3
8         where
9           inheriting conditions from entity ZI_RAP_TRAVEL_tm3;
10 }
11

```

Figura 20: Definição das regras de acesso para a visão de projeção do BO viagem.

A próxima fase é ativar os recursos transacionais da nossa aplicação, criando o *behavior definition* do business object: neste teremos a definição básica do comportamento, que define o nosso business object das viagens *managed*<sub>21</sub> com o criar, atualizar e apagar, bem como as associações necessárias. É muito importante que o nome do *behavior definition* tenha o mesmo nome da root view do CDS. Neste anexo apenas estou a implementar a aplicação do tipo managed.

A primeira coisa que definimos é o alias (figura 21) para a viagem e a entidade de reserva. Depois, especificamos a persistência dando os nomes das tabelas da base de dados para ambas as entidades. Isso permite o runtime managed de execução para executar as operações criar, alterar e apagar diretamente nas nossas tabelas de base de dados. O próximo passo é especificar o master lock para a entidade root. Para isso, adicionamos a linha 8 "*lock master*<sub>22</sub>". O nó “filho” da reserva torna-se dependente do lock e faz uso da associação definida na CDS view para permitir ao transacional fazer a associação de viagens explicitamente listadas na entidade de reserva. Como este é um cenário baseado em UUID, queremos que o runtime managed forneça uma chave quando novas instâncias são criadas. Para isso, precisamos de

fazer a numeração TravelUUID managed e somente de leitura. O mesmo é necessário para a entidade de reserva.

```

@TRL ZI_RAP_TRAVEL_TM3
1 managed;
2 with draft;
3
4@define behavior for ZI_RAP_Travel_tm3 alias Travel
5 implementation in class zbp_i_rap_travel_tm3 unique
6 persistent table zrap_atrav_tm3
7 draft table zrap_dtrav_tm3
8 lock master total etag LastChangedAt
9 authorization master ( instance )
10 etag master LocalLastChangedAt
11 {
12   create;
13   update;
14   delete;
15   association _Booking { create; with draft; }
16
17   field ( numbering : managed, readonly ) TravelUUID;
18   field ( readonly ) TravelID, TotalPrice, TravelStatus;
19   field ( readonly ) LastChangedAt, LastChangedBy, CreatedAt, CreatedBy, LocalLastChangedAt;
20   field ( mandatory ) AgencyID, CustomerID;
21
22   action ( features : instance ) acceptTravel result [1] $self;
23   action ( features : instance ) rejectTravel result [1] $self;
24   internal action recalcTotalPrice;
25
26   determination setInitialStatus on modify { create; }
27   determination calculateTotalPrice on modify { field BookingFee, CurrencyCode; }
28   determination calculateTravelID on save { create; }
29
30   validation validateAgency on save { field AgencyID; create; }
31   validation validateCustomer on save { field CustomerID; create; }
32   validation validateDates on save { field BeginDate, EndDate; create; }
33
34   draft determine action Prepare {
35     validation validateAgency;
36     validation validateCustomer;
37     validation validateDates;
38   }
39
40@ mapping for zrap_atrav_tm3
41 {
42   TravelUUID = travel_uuid;
43   TravelID = travel_id;
44   AgencyID = agency_id;
45   CustomerID = customer_id;
46   BeginDate = begin_date;
47   EndDate = end_date;
48   BookingFee = booking_fee;
49   TotalPrice = total_price;
50   CurrencyCode = currency_code;
51   Description = description;
52   TravelStatus = overall_status;
53   CreatedBy = created_by;
54   CreatedAt = created_at;
55   LastChangedBy = last_changed_by;
56   LastChangedAt = last_changed_at;
57   LocalLastChangedAt = local_last_changed_at;
58 }
59 }
60
61@define behavior for ZI_RAP_Booking_tm3 alias Booking
62 implementation in class zbp_i_rap_booking_tm3 unique
63 persistent table zrap_abook_tm3
64 draft table zrap_dbook_tm3
65 lock dependent by _Travel
66 authorization dependent by _Travel
67 etag master LocalLastChangedAt
68 {
69   update;
70   delete;
71
72   association _Travel { with draft; }
73
74   field ( numbering : managed, readonly ) BookingUUID;
75   field ( readonly ) TravelUUID, BookingID;
76   field ( readonly ) CreatedBy, LastChangedBy, LocalLastChangedAt;
77
78   determination calculateBookingID on modify { create; }
79   determination calculateTotalPrice on modify { field FlightPrice, CurrencyCode; }
80
81@ mapping for zrap_abook_tm3
82 {
83   BookingUUID = booking_uuid;
84   TravelUUID = travel_uuid;
85   BookingID = booking_id;
86   BookingDate = booking_date;
87   CustomerID = customer_id;
88   CarrierID = carrier_id;
89   ConnectionID = connection_id;
90   FlightDate = flight_date;
91   FlightPrice = flight_price;
92   CurrencyCode = currency_code;
93   CreatedBy = created_by;
94   LastChangedBy = last_changed_by;
95   LocalLastChangedAt = local_last_changed_at;
96 }
97 }

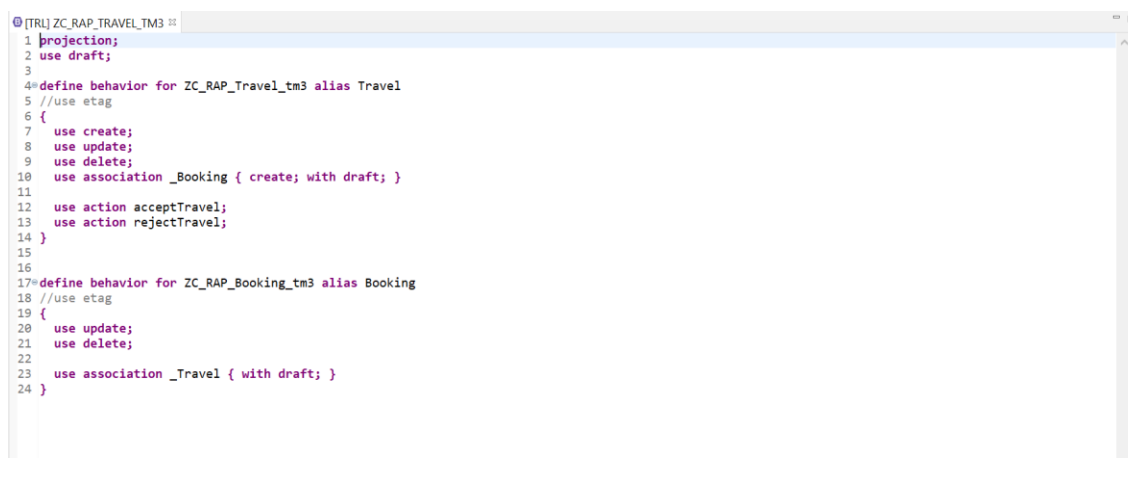
```

Figura 21: Ativar dos recursos transacionais.

Vai aparecer um aviso indicando que o TravelUUID na entidade de reserva deve ser definido para somente leitura, pois é usado na condição de associação. A solução para o problema é definir o master *Etag*<sub>23</sub> em ambas as entidades. O outro aviso que aparece diz respeito às informações de mapeamento que faltam. Como fornecemos aliases nas visualizações de CDS da interface para os nomes dos elementos, precisamos de dizer ao framework como mapear os nomes dos elementos no modelo de dados CDS para os campos da tabela correspondente. O mesmo precisa de ser feito para a entidade de reserva.

Passamos agora para o behavior definition (figura 22), que projeta os recursos transacionais da *behavior definition base*. O nome da behavior definition deve ser idêntico ao da visão raiz do CDS.

As operações de *create*, *update* e *delete* são automaticamente assumidas a partir do behavior definition base por meio da keyword "use". A primeira coisa que definimos é o *alias* para a entidade viagem e reserva. Nós também queremos permitir a manipulação de Etag e para isso precisamos de adicionar isso para todas as entidades.



```
1 projection;
2 use draft;
3
4 define behavior for ZC_RAP_Travel_tm3 alias Travel
5 //use etag
6 {
7   use create;
8   use update;
9   use delete;
10  use association _Booking { create; with draft; }
11
12  use action acceptTravel;
13  use action rejectTravel;
14 }
15
16
17 define behavior for ZC_RAP_Booking_tm3 alias Booking
18 //use etag
19 {
20   use update;
21   use delete;
22
23   use association _Travel { with draft; }
24 }
```

Figura 22: Behavior definition.

Relativamente à EML, é a linguagem de manipulação de entidades usadas para, por exemplo, adicionar determinações, validações ou ações ao behavior definition do business object. O EML padrão permite de forma segura read e modify o acesso a dados. Na criação desta funcionalidade no ADT é necessário adicionar a interface `if_oo_adt_classrun`. O valor uuid que aparece é retirado da tabela das viagens, porque funciona como key.

A seguir, conforme apresentado na figura 23, temos a primeira operação que vemos que é a operação de leitura, que permite o desempenho de uma leitura transacional de instâncias de business objects.

Leitura transacional significa que a leitura está a ocorrer no buffer transacional. Se a instância não está presente no buffer, ele é lido automaticamente no buffer da base de dados pelo runtime managed.

As operações de modify são usadas para realizar alterações no buffer transacional. Se um registo não estiver presente no buffer, ele é lido da base de dados antes da operação a ser executada.

A modify update é usada para atualizar instâncias, sendo possível, através do Content ID, também atualizar instâncias que já foram criadas antes, mas que ainda não foram mantidas na base de dados. A modify delete é usada para apagar instâncias.



TRL ZCL\_RAP\_EML\_TM3

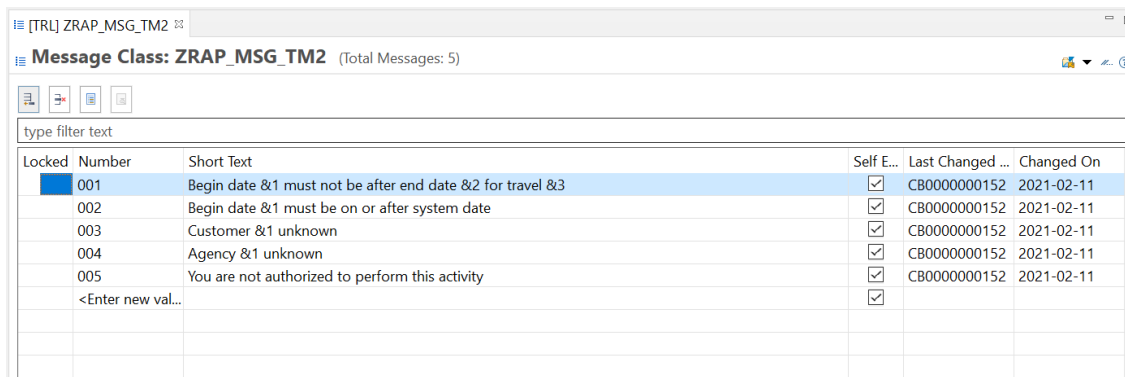
© [TRL] ZCL\_RAP\_EML\_TM3 33

Figura 22: Clases de ejemplo de como se funcionan las funciones

Figura 23: Classe de exemplo de como as funcionalidades funcionam.

Na implementação do behavior, vamos usar as nossas próprias mensagens através de uma classe de exceção, “T100” geradas. Por isso, primeiro precisamos de criar uma classe para as nossas mensagens. Na criação é preciso ter em conta que a superclasse precisa de ser CX\_STATIC\_CHECK, e, na parte pública devemos adicionar a interface if\_abap\_behv\_message.

Vamos criar cinco mensagens de erro, por exemplo com os números de um a cinco, adicionando as constantes para todas as cinco mensagens. O *behavior implementation* acontece nas *Local Types tab* geradas pela tool numa classe herdada, designada por *cl\_abap\_behavior\_handler*. Na parte privada do código, adicionamos constantes para o status da viagem.



Message Class: ZRAP\_MSG\_TM2 (Total Messages: 5)

Locked	Number	Short Text	Self E...	Last Changed ...	Changed On
	001	Begin date &1 must not be after end date &2 for travel &3	<input checked="" type="checkbox"/>	CB0000000152	2021-02-11
	002	Begin date &1 must be on or after system date	<input checked="" type="checkbox"/>	CB0000000152	2021-02-11
	003	Customer &1 unknown	<input checked="" type="checkbox"/>	CB0000000152	2021-02-11
	004	Agency &1 unknown	<input checked="" type="checkbox"/>	CB0000000152	2021-02-11
	005	You are not authorized to perform this activity	<input checked="" type="checkbox"/>	CB0000000152	2021-02-11
	<Enter new val...		<input checked="" type="checkbox"/>		

Figura 24: Message Class. Classe das mensagens de erro.

```

1 CLASS zcm_rap_tm3 DEFINITION
2 PUBLIC
3 INHERITING FROM cx_static_check
4 FINAL
5 CREATE PUBLIC .
6
7 PUBLIC SECTION.
8
9 INTERFACES if_t100_dyn_msg .
10 INTERFACES if_t100_message .
11 INTERFACES if_abap_behv_message.
12
13 CONSTANTS:
14 BEGIN OF date_interval,
15   msgid TYPE symmsgid VALUE 'ZRAP_MSG_tm3',
16   msgno TYPE symsgno VALUE '001',
17   attr1 TYPE scx_attrname VALUE 'BEGINDATE',
18   attr2 TYPE scx_attrname VALUE 'ENDDATE',
19   attr3 TYPE scx_attrname VALUE 'TRAVELID',
20   attr4 TYPE scx_attrname VALUE '',
21 END OF date_interval .
22
23 CONSTANTS:
24 BEGIN OF begin_date_before_system_date,
25   msgid TYPE symmsgid VALUE 'ZRAP_MSG_tm3',
26   msgno TYPE symsgno VALUE '002',
27   attr1 TYPE scx_attrname VALUE 'BEGINDATE',
28   attr2 TYPE scx_attrname VALUE '',
29   attr3 TYPE scx_attrname VALUE '',
30   attr4 TYPE scx_attrname VALUE '',
31 END OF begin_date_before_system_date .
32
33 CONSTANTS:
34 BEGIN OF customer_unknown,
35   msgid TYPE symmsgid VALUE 'ZRAP_MSG_tm3',
36   msgno TYPE symsgno VALUE '003',
37   attr1 TYPE scx_attrname VALUE 'CUSTOMERID',
38   attr2 TYPE scx_attrname VALUE '',
39   attr3 TYPE scx_attrname VALUE '',
40   attr4 TYPE scx_attrname VALUE '',
41 END OF customer_unknown .
42
43 CONSTANTS:
44 BEGIN OF agency_unknown,
45   msgid TYPE symmsgid VALUE 'ZRAP_MSG_tm3',
46   msgno TYPE symsgno VALUE '004',
47   attr1 TYPE scx_attrname VALUE 'AGENCYID',
48   attr2 TYPE scx_attrname VALUE '',
49   attr3 TYPE scx_attrname VALUE '',
50   attr4 TYPE scx_attrname VALUE '',
51 END OF agency_unknown .
52
53 CONSTANTS:
54 BEGIN OF you_are_not_authorized,
55   msgid TYPE symmsgid VALUE 'ZRAP_MSG_tm3',
56   msgno TYPE symsgno VALUE '005',
57   attr1 TYPE scx_attrname VALUE 'ACTIVITY',
58   attr2 TYPE scx_attrname VALUE '',
59   attr3 TYPE scx_attrname VALUE '',
60   attr4 TYPE scx_attrname VALUE '',
61 END OF you_are_not_authorized .
62
63 ENDCLASS.

```

```

39 [TRL] ZCM_RAP_TM3
40 ENH UP customer_unknown .
41 CONSTANTS:
42 BEGIN OF agency_unknown,
43   msgid TYPE symsgid VALUE 'ZRAP_MSG_tm3',
44   msgno TYPE symsgno VALUE '004',
45   attr1 TYPE scx_attrname VALUE 'AGENCYID',
46   attr2 TYPE scx_attrname VALUE '',
47   attr3 TYPE scx_attrname VALUE '',
48   attr4 TYPE scx_attrname VALUE '',
49 END OF agency_unknown .
50 CONSTANTS:
51 BEGIN OF unauthorized,
52   msgid TYPE symsgid VALUE 'ZRAP_MSG_tm3',
53   msgno TYPE symsgno VALUE '005',
54   attr1 TYPE scx_attrname VALUE '',
55   attr2 TYPE scx_attrname VALUE '',
56   attr3 TYPE scx_attrname VALUE '',
57   attr4 TYPE scx_attrname VALUE '',
58 END OF unauthorized .
59 METHODS constructor
60 IMPORTING
61   severity TYPE if_abap_behv_message=>t_severity DEFAULT if_abap_behv_message=>severity-error
62   textid LIKE if_t100_message=>t100key OPTIONAL
63   previous TYPE REF TO cx_root OPTIONAL
64   begindate TYPE /dmo/begin_date OPTIONAL
65   enddate TYPE /dmo/end_date OPTIONAL
66   travelid TYPE /dmo/travel_id OPTIONAL
67   customerid TYPE /dmo/customer_id OPTIONAL
68   agencycid TYPE /dmo/agency_id OPTIONAL
69 .
70
71 DATA begindate TYPE /dmo/begin_date READ-ONLY.
72 DATA enddate TYPE /dmo/end_date READ-ONLY.
73 DATA travelid TYPE string READ-ONLY.
74 DATA customerid TYPE string READ-ONLY.
75 DATA agencycid TYPE string READ-ONLY.
76
77 PROTECTED SECTION.
78 PRIVATE SECTION.
79 ENDCLASS.
80
81

```

Figura 24: Classe que permite o display dessas mensagens de erro.

## 7º : Classe de implementação dos métodos que representam as funcionalidades na aplicação

```

3 [TRL] ZBP_I_RAP_TRAVEL_TM3
4
5 1= CLASS lhc_Travel DEFINITION INHERITING FROM c1_abap_behavior_handler.
6   PRIVATE SECTION.
7
8   CONSTANTS:
9     BEGIN OF travel_status,
10      open TYPE c LENGTH 1 VALUE 'O', " Open
11      accepted TYPE c LENGTH 1 VALUE 'A', " Accepted
12      canceled TYPE c LENGTH 1 VALUE 'X', " Cancelled
13     END OF travel_status.
14
15   METHODS CalculateTravelID FOR DETERMINE ON SAVE
16     IMPORTING keys FOR travel-CalculateTravelID.
17
18   METHODS setInitialStatus FOR DETERMINE ON MODIFY
19     IMPORTING keys FOR Travel-setInitialStatus.
20
21   METHODS validateAgency FOR VALIDATE ON SAVE
22     IMPORTING keys FOR Travel-validateAgency.
23
24   METHODS validatecustomer FOR VALIDATE ON SAVE
25     IMPORTING keys FOR travel-validatecustomer.
26
27   METHODS validateDates FOR VALIDATE ON SAVE
28     IMPORTING keys FOR Travel-validateDates.
29
30   METHODS acceptTravel FOR MODIFY
31     IMPORTING keys FOR ACTION Travel-acceptTravel RESULT result.
32
33   METHODS rejectTravel FOR MODIFY
34     IMPORTING keys FOR ACTION Travel-rejectTravel RESULT result.
35
36   METHODS get_features FOR FEATURES
37     IMPORTING keys REQUEST requested_features FOR Travel RESULT result.
38
39   METHODS get_authorizations FOR AUTHORIZATION
40     IMPORTING keys REQUEST requested_authorizations FOR Travel RESULT result.
41
42   METHODS recalcTotalPrice FOR MODIFY
43     IMPORTING keys FOR ACTION travel-recalcTotalPrice.
44
45   METHODS calculateTotalPrice FOR DETERMINE ON MODIFY
46     IMPORTING keys FOR Travel-calculateTotalPrice.
47
48   METHODS is_update_granted IMPORTING has_before_image TYPE abap_bool
49     overall_status TYPE /dmo/overall_status
50     RETURNING VALUE(update_granted) TYPE abap_bool.
51
52   METHODS is_delete_granted IMPORTING has_before_image TYPE abap_bool
53     overall_status TYPE /dmo/overall_status
54     RETURNING VALUE(delete_granted) TYPE abap_bool.
55
56   RETURN is_cancelled RETURNING VALUE(cancelled) TYPE abap_bool.
57

```

Figura 25: Classe de implementação dos métodos.

O primeiro método (figura 26) que temos que implementar é o método `acceptTravel`, que define se o status é aceite para todas as *keys* fornecidas usando uma instrução de modificação EML. Ele usa "in local mode", porque, por exemplo, é o que nos permite até mesmo mudar campos apenas de leitura enquanto saltamos o recurso e o controle de autorização.

Uma pequena nota: o uso de `%tky` significa chave de transação. No caso de a usarmos sem a funcionalidade *draft*, consideramos o mesmo valor da chave percentual, que é a chave da entidade relacionada. O uso da chave transacional reduz a necessidade para voltar a desenvolver a implementação quando permite por exemplo o rascunho, e é semelhante à ação de aceitar a viagem ou de a rejeitar. Agora a chave transacional vai ter automaticamente o indicador `is_draft`.

```

246
247 METHOD acceptTravel.
248   " Set the new overall status
249   MODIFY ENTITIES OF zi_rap_travel_tm3 IN LOCAL MODE
250   ENTITY Travel
251   UPDATE
252     FIELDS ( TravelStatus )
253     WITH VALUE #( FOR key IN keys
254                   ( %tky = key-%tky
255                     TravelStatus = travel_status-accepted ) )
256   FAILED failed
257   REPORTED reported.
258
259   " Fill the response table
260   READ ENTITIES OF zi_rap_travel_tm3 IN LOCAL MODE
261   ENTITY Travel
262     ALL FIELDS WITH CORRESPONDING #( keys )
263   RESULT DATA(travels).
264
265   result = VALUE #( FOR travel IN travels
266                     ( %tky = travel-%tky
267                       %param = travel ) ).
268   ENDMETHOD.
269

```

Figura 26: Método `acceptTravel`.

A primeira validação que vamos implementar é o método `validateAgency` (figura 27): esta validação verifica se o `AgencyID` fornecido ao guardar, ou quando uma instância é criada, foi alterada. Validações são implementações que geralmente começam com a leitura dos dados necessários usando EML. No nosso caso, queremos ler o ID da agência para as chaves fornecidas.

```

120
121 METHOD validateAgency.
122   " Read relevant travel instance data
123   READ ENTITIES OF zi_rap_travel_tm3 IN LOCAL MODE
124   ENTITY Travel
125     FIELDS ( AgencyID ) WITH CORRESPONDING #( keys )
126   RESULT DATA(travels).
127
128   DATA agencies TYPE SORTED TABLE OF /dmo/agency WITH UNIQUE KEY agency_id.
129
130   " Optimization of DB select: extract distinct non-initial agency IDs
131   agencies = CORRESPONDING #( travels DISCARDING DUPLICATES MAPPING agency_id = AgencyID EXCEPT * ).
132   DELETE agencies WHERE agency_id IS INITIAL.
133
134 IF agencies IS NOT INITIAL.
135   " Check if agency ID exist
136   SELECT FROM /dmo/agency FIELDS agency_id
137   FOR ALL ENTRIES IN agencies
138   WHERE agency_id = @agencies-agency_id
139   INTO TABLE @DATA(agencies_db).
140 ENDIF.
141
142   " Raise msg for non existing and initial agencyID
143 LOOP AT travels INTO DATA(travel).
144   " Clear state messages that might exist
145   APPEND VALUE #( %tky = travel-%tky
146                   %state_area = 'VALIDATE_AGENCY' )
147   TO reported-travel.
148
149 IF travel-AgencyID IS INITIAL OR NOT line_exists( agencies_db[ agency_id = travel-AgencyID ] ).
150   APPEND VALUE #( %tky = travel-%tky ) TO failed-travel.
151
152   APPEND VALUE #( %tky = travel-%tky
153                   %state_area = 'VALIDATE_AGENCY'
154                   %msg = NEW zcm_rap_tm3(
155                     severity = if_abap_behv_message>severity-error
156                     textid = zcm_rap_tm3>agency_unknown
157                     agencyid = travel-AgencyID )
158                   %element-AgencyID = if_abap_behv>%mk-on )
159   TO reported-travel.
160 ENDIF.
161 ENDOLOOP.
162 ENDMETHOD.
163

```

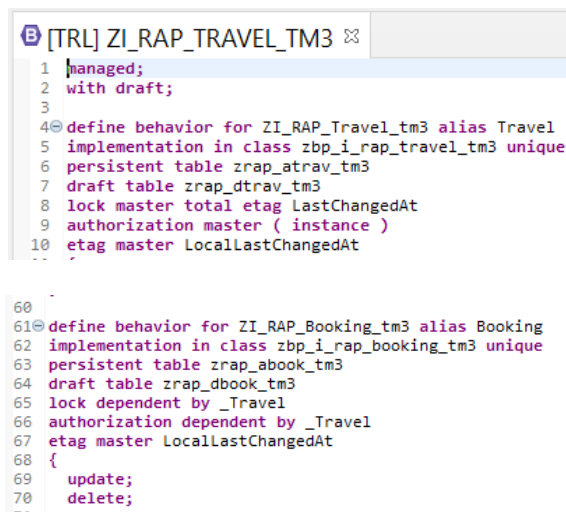
Figura 27: Método `validateAgency`.

Foi criada uma tabela interna com todos os identificadores das diversas agências e foi feito um select da base de dados para confirmar que existem, verificando se um *AgencyID* foi fornecido e se está ativo. Se o *AgencyID* estiver vazio ou não existe na tabela, inserimos uma mensagem usando a nossa classe de exceção.

Uma nota para as determinações que precisam de ser idempotentes, isto é o resultado mantém-se sendo executadas várias vezes para a mesma key.

De seguida, vamos adicionar controle de autorização ao nosso *business object*.

A linha 8 "authorization master (instance)", é o que declara o nó raiz como authorization master. Relativamente à parte do *booking* fazemos a mesma mudança mais a especificação à associação *\_Travel*. A *authorization master* diz que a entidade de reserva é dependente da autorização. Vão aparecer uns aviso que, usando a correção rápida, gera automaticamente as implementações para todos os três métodos.



```
[TRL] ZI_RAP_TRAVEL_TM3 ⌕
1 managed;
2 with draft;
3
4 define behavior for ZI_RAP_Travel_tm3 alias Travel
5 implementation in class zbp_i_rap_travel_tm3 unique
6 persistent table zrap_atrav_tm3
7 draft table zrap_dtrav_tm3
8 lock master total etag LastChangedAt
9 authorization master ( instance )
10 etag master LocalLastChangedAt
11
60
61 define behavior for ZI_RAP_Booking_tm3 alias Booking
62 implementation in class zbp_i_rap_booking_tm3 unique
63 persistent table zrap_abook_tm3
64 draft table zrap_dbook_tm3
65 lock dependent by _Travel
66 authorization dependent by _Travel
67 etag master LocalLastChangedAt
68 {
69   update;
70   delete;
71
```

Figura 26: Controlo de autorização.

Com desenvolvimento destas funcionalidades resultou a aplicação apresentada nos anexos do relatório.

## 8º : Comparação entre a solução SAP Abap RESTfull programming model e Oracle Apex

O Oracle Apex foi uma ferramenta de desenvolvimento de base de dados e aplicações web que se utiliza na cadeira de base de dados da NOVA FCT. O oracle Apex tem uma boa velocidade de desenvolvimento de páginas web, a criação por meio de wizards facilita muito a vida do programador, a integração entre base de dados e front-end é extremamente simples. Para além disso, o APEX é um sistema muito bem documentado. Para novos programadores no APEX é bastante positivo a sua integração, devido à facilidade de interação com a interface web, utilizada para gerir todas as camadas da aplicação. Por outro lado, a interface da aplicação é muito pouco flexível e bastante ilimitada, dificultando a personalização do front-end.

```

1 drop table travel;
2
3 create table travel (uuid_t number primary key, price_t number);
4 insert into travel values (1,1);
5 insert into travel values (2,1);
6 insert into travel values (3,1);
7 insert into travel values (4,1);
8 insert into travel values (5,1);
9
10 drop table booking;
11 create table booking(uuid_b number, price number, uuid_t number);
12 insert into booking values (10,500,1);
13 insert into booking values (20,500,2);
14 insert into booking values (30,500,3);
15
16 drop view total_price;
17 create view total_price as select sum(booking.price) price_t, uuid_t from travel inner join booking using(uuid_t) group by uuid_t;
18
19 insert into booking values (20,500,1);
20 select * from total_price;
21
22 create or replace trigger calc_total_price
23 instead of update on total_price
24 for each row
25 begin
26     update travel
27     set price_t= (select max(price_t)+ :new.price_t
28                  from total_price
29                  where uuid_t=:new.uuid_t)
30     where uuid_t = :new.uuid_t;
31 end;
32 /
33
34 select * from travel;
35 update total_price set price_t = 95 where uuid_t=1;
36 select * from travel;
37
38

```

*Listagem 1: Método total\_price em SQL. Este método soma o preço dos bookings todos associados a uma viagem mais o custo da própria viagem.*

Estas duas aplicações são ambas muito boas. Comparando algumas funcionalidades vamos poder ver quais são as mais fáceis ou para a SAP Abap RESTfull programming model ou para o Oracle Apex. A primeira funcionalidade a apresentar é por exemplo calcular o preço total da viagem (incluindo as reservas).

Em SQL existem as tableas booking e travel e podemos criar uma vista e também triggers.

Relativamente à SAP Abap RESTfull programming model teremos um método que vai ler de todas as viagens para a reserva feita e se houver várias reservas na mesma viagem, apenas uma é devolvida. Depois criamos um trigger também que calcula o preço total.

```

455@ METHOD recalctotalprice.
456
457 TYPES: BEGIN OF ty_amount_per_currencycode,
458         amount          TYPE /dmo/total_price,
459         currency_code    TYPE /dmo/currency_code,
460     END OF ty_amount_per_currencycode.
461
462 DATA: amount_per_currencycode TYPE STANDARD TABLE OF ty_amount_per_currencycode.
463
464 " Read all relevant travel instances.
465 READ ENTITIES OF zi_rap_travel_tm3 IN LOCAL MODE
466     ENTITY Travel
467     FIELDS ( BookingFee CurrencyCode )
468     WITH CORRESPONDING #( keys )
469     RESULT DATA(travels).
470
471 DELETE travels WHERE CurrencyCode IS INITIAL.
472
473@ LOOP AT travels ASSIGNING FIELD-SYMBOL(<travel>).
474     " Set the start for the calculation by adding the booking fee.
475     amount_per_currencycode = VALUE #( ( amount          = <travel>-BookingFee
476                                         currency_code    = <travel>-CurrencyCode ) ).
477
478     " Read all associated bookings and add them to the total price.
479     READ ENTITIES OF ZI_RAP_Travel_tm3 IN LOCAL MODE
480         ENTITY Travel BY \Booking
481         FIELDS ( FlightPrice CurrencyCode )
482         WITH VALUE #( ( %tky = <travel>-%tky ) )
483         RESULT DATA(bookings).
484
485@ LOOP AT bookings INTO DATA(booking) WHERE CurrencyCode IS NOT INITIAL.
486     COLLECT VALUE ty_amount_per_currencycode( amount          = booking-FlightPrice
487                                                currency_code    = booking-CurrencyCode ) INTO amount_per_currencycode.
488
489 ENDLOOP.
490
491@ CLEAR <travel>-TotalPrice.
492 LOOP AT amount_per_currencycode INTO DATA(single_amount_per_currencycode).
493     " If needed do a Currency Conversion
494     IF single_amount_per_currencycode-currency_code = <travel>-CurrencyCode.
495         <travel>-TotalPrice += single_amount_per_currencycode-amount.
496     ELSE.
497         /dmo/cl_flight_amdp=>convert_currency(
498             EXPORTING
499                 iv_amount          = single_amount_per_currencycode-amount
500                 iv_currency_code_source = single_amount_per_currencycode-currency_code
501                 iv_currency_code_target  = <travel>-CurrencyCode
502                 iv_exchange_rate_date   = cl_abap_context_info=>get_system_date( )
503             IMPORTING
504                 ev_amount          = DATA(total_booking_price_per_curr)
505             ).
506         <travel>-TotalPrice += total_booking_price_per_curr.
507     ENDIF.
508 ENDLOOP.
509
510 " write back the modified total_price of travels
511 MODIFY ENTITIES OF ZI_RAP_Travel_tm3 IN LOCAL MODE
512     ENTITY travel
513     UPDATE FIELDS ( TotalPrice )
514     WITH CORRESPONDING #( travels ).
515
516 ENDMETHOD.
517
518@ METHOD calculateTotalPrice.
519
520 MODIFY ENTITIES OF zi_rap_travel_tm3 IN LOCAL MODE
521     ENTITY travel
522     EXECUTE recalctotalprice
523     FROM CORRESPONDING #( keys )
524     REPORTED DATA(execute_reported).
525
526     reported = CORRESPONDING #( DEEP execute_reported ).
527
528 ENDMETHOD.
529

```

*Listagem 2: Método total\_price em ABAP. Este método soma o preço dos bookings todos associados a uma viagem mais o custo da própria viagem.*

Comparando ambas soluções, em SQL se apenas fosse necessário criar sem atualizar a solução APEX seria mais declarativa e potencial menos complexa. No entanto, como é possível apagar bookings e por consequência alterar o preço total, em ORACLE APEX seria ainda necessário criar triggers para todos os casos possíveis (insert, update e delete). Existem assim a possibilidade desta solução ficar mais

complexa relativamente à solução mais imperativa do ABAP. Para além disso, a nível de interface também é mais complicado na integração de diversos campos, de diversas entidades.

Outra funcionalidade a apresentar é o *value help*. No APEX também é possível fazer, no entanto faz de forma muito diferente. No APEX é referido como *field level help*<sup>27</sup> e contém informações relacionadas com um campo específico, dependendo do campo específico, o sistema mostra um dos seguintes tipos de ajuda:

- Janela de pesquisa;
- Lista de valores válidos;
- Explicação de campo;

Neste caso, no APEX esta funcionalidade não tem o mesmo efeito que no ABAP. No glossário do *field level help*<sup>27</sup> está um link de referência de uma melhor e mais detalhada explicação.

Relativamente à SAP Abap RESTfull programming model é feita através de uma anotação apenas:

```
@Consumption.valueHelpDefinition: [{ entity: { name: '/DMO/I_Agency', element: 'AgencyID' } }]
```

Contudo será necessário saber ou fazer alguma pesquisa na documentação para se saber que é esta a anotação a ser usada.

Uma última funcionalidade a apresentar é o *draft*, que no APEX chama-se *savepoint*<sup>29</sup>. No APEX define-se um momento que dentro de uma transação, a qualquer momento, podemos reverter o caminho depois desse *savepoint*. Estes são apenas suportados pelas transações locais, o que pode não ser muito prático. O *savepoint* é criado através de um método `Connection.setSavepoint` que retorna uma instância `java.sql.Savepoint`. Depois especifica-se o nome do *savepoint* como parâmetro do tipo *String* do método `setSavepoint`. Como não é obrigatório especificar o nome, se não o fizer o *savepoint* assume um ID. No glossário do *savepoint*<sup>26</sup> está um link de referência de uma melhor e mais detalhada explicação.

A forma como a SAP Abap RESTfull programming model ativa esta funcionalidade é adicionando as seguintes linhas/pedaços de código às seguintes classes:

1º - A classe `ZI_RAP_TRAVEL_tm` no *behavior definition*:

Nesta classe vamos especificar a funcionalidade *draft* para cada entidade:

```
1 managed;
2 with draft;
3
4 define behavior for ZI_RAP_Travel_tm3 alias Travel
5 implementation in class zbp_i_rap_travel_tm3 unique
6 persistent table zrap_atrav_tm3
7 draft table zrap_dtrav_tm3
8 lock master total etag LastChangedAt
9 authorization master ( instance )
10 etag master LocalLastChangedAt
```

...

```
15 association _Booking { create; with draft; }
16
```



```

...
64 draft table zrap_dbook_tm3
...
71
72 association _Travel { with draft; }
73

```

Listagem 3: Funcionalidade draft em ABAP

2º - Depois cria-se a tabela *draft*, ou seja a tabela que vai ter as viagens e as reservas como rascunho. Esta tabela nova criada é referida no código acima como `draft table zrap_dtrav_tm` para a viagem e `draft table zrap_dbook_tm` para as reservas.

3º - A seguir, o *draft* ainda não tem nenhuma projeção por isso é necessário criar na classe `ZC_RAP_TRAVEL_tm`:

```

...
2 use draft;
3
...
10 use association _Booking { create; with draft; }
11
...
23 use association _Travel { with draft; }

```

Listagem 4: Funcionalidade draft em ABAP, projection.

4º -

Finalmente na classe principal `ZBP_I_RAP_TRAVEL` no método `get_authorizations`, aqui a funcionalidade adiciona duas ações predefinidas de preparação e de edição da funcionalidade, ao *business object* (primeiro retângulo) e é o que permite tornar mais um controle de autorização.

```

320 METHOD get_authorizations.
321 DATA: has_before_image TYPE abap_bool,
322        is_update_requested TYPE abap_bool,
323        is_delete_requested TYPE abap_bool,
324        update_granted TYPE abap_bool,
325        delete_granted TYPE abap_bool.
326
327 DATA: failed_travel LIKE LINE OF failed-travel.
328
329 " Read the existing travels
330 READ ENTITIES OF zi_rap_travel_tm3 IN LOCAL MODE
331 ENTITY Travel
332 FIELDS ( TravelStatus ) WITH CORRESPONDING #( keys )
333 RESULT DATA(travels)
334 FAILED failed.
335
336 CHECK travels IS NOT INITIAL.
337
338 * In this example the authorization is defined based on the Activity + Travel Status
339 * For the Travel Status we need the before-image from the database. We perform this for active (is_draft=00)
340 * as well as for drafts (is_draft=01) as we can't distinguish between edit or new drafts
341 SELECT FROM zrap_atrav_tm3
342 FIELDS travel_uuid,overall_status
343 FOR ALL ENTRIES IN @travels
344 WHERE travel_uuid EQ @travels-TravelUUID
345 ORDER BY PRIMARY KEY
346 INTO TABLE @DATA(travels_before_image).
347
348 is_update_requested = COND #( WHEN requested_authorizations-%update = if_abap_behv=>mk-on OR
349                                requested_authorizations-%action-acceptTravel = if_abap_behv=>mk-on OR
350                                requested_authorizations-%action-rejectTravel = if_abap_behv=>mk-on OR
351                                requested_authorizations-%action-Prepare = if_abap_behv=>mk-on OR
352                                requested_authorizations-%action-Edit = if_abap_behv=>mk-on OR
353                                requested_authorizations-%assoc-Booking = if_abap_behv=>mk-on
354                                THEN abap_true ELSE abap_false ).
355
356 is_delete_requested = COND #( WHEN requested_authorizations-%delete = if_abap_behv=>mk-on
357                                THEN abap_true ELSE abap_false ).
358
359 LOOP AT travels INTO DATA(travel).
360   update_granted = delete_granted = abap_false.
361
362   READ TABLE travels_before_image INTO DATA(travel_before_image)
363   WITH KEY travel_uuid = travel-TravelUUID BINARY SEARCH.
364   has_before_image = COND #( WHEN sy-subrc = 0 THEN abap_true ELSE abap_false ).
365
366   IF is_update_requested = abap_true.
367     " Edit of an existing record -> check update authorization
368     IF has_before_image = abap_true.
369       update_granted = is_update_granted( has_before_image = has_before_image overall_status = travel_before_image-overall_status ).
370     IF update_granted = abap_false.
371       APPEND VALUE #( %tky = travel-%tky
372                      %msg = NEW zcm_rap_tm3( severity = if_abap_behv_message=>severity-error
373                                              textid = zcm_rap_tm3=>unauthorized )
374                      ) TO reported-travel.
375     ENDIF.
376     " Creation of a new record -> check create authorization
377   ELSE.
378     update_granted = is_create_granted( ).
379   IF update_granted = abap_false.
380     APPEND VALUE #( %tky = travel-%tky
381                    %msg = NEW zcm_rap_tm3( severity = if_abap_behv_message=>severity-error
382                                            textid = zcm_rap_tm3=>unauthorized )
383                    ) TO reported-travel.
384   ENDIF.
385 ENDIF.
386 ENDIF.
387
388   IF is_delete_requested = abap_true.
389     delete_granted = is_delete_granted( has_before_image = has_before_image overall_status = travel_before_image-overall_status ).
390   IF delete_granted = abap_false.
391     APPEND VALUE #( %tky = travel-%tky
392                    %msg = NEW zcm_rap_tm3( severity = if_abap_behv_message=>severity-error
393                                            textid = zcm_rap_tm3=>unauthorized )
394                    ) TO reported-travel.
395   ENDIF.
396 ENDIF.

```

```

387
388
389 IF is_delete_requested = abap_true.
390   delete_granted = is_delete_granted( has_before_image = has_before_image overall_status = travel_before_image-overall_status ).
391   IF delete_granted = abap_false.
392     APPEND VALUE #( %tky          = travel-%tky
393                     %msg          = NEW zcm_rap_tm3( severity = if_abap_behv_message=>severity-error
394                                                         textid   = zcm_rap_tm3=>unauthorized )
395                     ) TO reported-travel.
396   ENDIF.
397 ENDIF.
398 APPEND VALUE #( %tky = travel-%tky
399
400                 %update          = COND #( WHEN update_granted = abap_true THEN if_abap_behv=>auth-allowed ELSE if_abap_behv=>auth-unauthorized )
401                 %action-acceptTravel = COND #( WHEN update_granted = abap_true THEN if_abap_behv=>auth-allowed ELSE if_abap_behv=>auth-unauthorized )
402                 %action-rejectTravel = COND #( WHEN update_granted = abap_true THEN if_abap_behv=>auth-allowed ELSE if_abap_behv=>auth-unauthorized )
403                 %action-Prepare      = COND #( WHEN update_granted = abap_true THEN if_abap_behv=>auth-allowed ELSE if_abap_behv=>auth-unauthorized )
404                 %action-Edit          = COND #( WHEN update_granted = abap_true THEN if_abap_behv=>auth-allowed ELSE if_abap_behv=>auth-unauthorized )
405                 %assoc_Booking        = COND #( WHEN update_granted = abap_true THEN if_abap_behv=>auth-allowed ELSE if_abap_behv=>auth-unauthorized )
406
407                 %delete            = COND #( WHEN delete_granted = abap_true THEN if_abap_behv=>auth-allowed ELSE if_abap_behv=>auth-unauthorized )
408
409             )
410   TO result.
411 ENDLOOP.
412 ENDMETHOD.
413

```

Listagem 5: Funcionalidade draft em ABAP, código.

## Conclusão

Neste anexo foi feita uma análise prática de como se faz uma aplicação SAP ABAP RESTfull programming model. Relativamente à configuração do ambiente foi utilizada uma licença trial com a duração de um mês, tendo-se verificado que para projectos longos o sistema pode levar à perda do projecto desenvolvido, pois não é possível guardar noutro meio. Na criação das tabelas utilizaram-se dados de demonstração o que para um exemplo de aplicação facilita a criação das mesmas.

O modelo de dados core data services (cds) é o que permite definir a estrutura da aplicação, pois é a base da mesma, preparando-a para os próximos passos, como é o caso do Odata. Na definição do Odata faz-se a ligação entre o eclipse e a SAP Fiori Web.

Para finalizar, na classe de implementação dos métodos observam-se as implementações das funcionalidades na aplicação, que é o que corresponde ao *back-end*. A ABAP RESTfull programming model é uma solução que tem muito potencial e a característica que sobressai é a simplicidade relativamente ao APEX, e que apesar de ainda ser recente tem bons pontos a adotar. É uma solução que para usufruir de todas as suas potencialidades requer já alguns conhecimentos da linguagem ABAP.

## Anexos

[TRL] ZI\_RAP\_TRAVEL\_TM3

quarta-feira, 3 de março de 2021, 17:54

```
1managed;
2with draft;
3
4define behavior for ZI_RAP_Travel_tm3 alias Travel
5implementation in class zbp_i_rap_travel_tm3 unique
6persistent table zrap_atrav_tm3
7draft table zrap_dtrav_tm3
8lock master total etag LastChangedAt
9authorization master ( instance )
10etag master LocalLastChangedAt
11{
12  create;
13  update;
14  delete;
15  association _Booking { create; with draft; }
16
17  field ( numbering : managed, readonly ) TravelUUID;
18  field ( readonly ) TravelId, TotalPrice, TravelStatus;
19  field ( readonly ) LastChangedAt, LastChangedBy, CreatedAt, CreatedBy, LocalLastChangedAt;
20  field ( mandatory ) AgencyID, CustomerID;
21
22  action ( features : instance ) acceptTravel result [1] $self;
23  action ( features : instance ) rejectTravel result [1] $self;
24  internal action recalcTotalPrice;
25
26  determination setInitialStatus on modify { create; }
27  determination calculateTotalPrice on modify { field BookingFee, CurrencyCode; }
28  determination calculateTravelID on save { create; }
29
30  validation validateAgency on save { field AgencyID; create; }
31  validation validateCustomer on save { field CustomerID; create; }
32  validation validateDates on save { field BeginDate, EndDate; create; }
33
34  draft determine action Prepare {
35    validation validateAgency;
36    validation validateCustomer;
37    validation validateDates;
38  }
39
40  mapping for zrap_atrav_tm3
41  {
42    TravelUUID = travel_uuid;
43    TravelID = travel_id;
44    AgencyID = agency_id;
45    CustomerID = customer_id;
46    BeginDate = begin_date;
47    EndDate = end_date;
48    BookingFee = booking_fee;
49    TotalPrice = total_price;
50    CurrencyCode = currency_code;
51    Description = description;
52    TravelStatus = overall_status;
53    CreatedBy = created_by;
54    CreatedAt = created_at;
55    LastChangedBy = last_changed_by;
56    LastChangedAt = last_changed_at;
57    LocalLastChangedAt = local_last_changed_at;
58  }
59}
60
61define behavior for ZI_RAP_Booking_tm3 alias Booking
62implementation in class zbp_i_rap_booking_tm3 unique
63persistent table zrap_abook_tm3
64draft table zrap_dbook_tm3
65lock dependent by _Travel
66authorization dependent by _Travel
67etag master LocalLastChangedAt
68{
69  update;
70  delete;
71
72  association _Travel { with draft; }
73
74  field ( numbering : managed, readonly ) BookingUUID;
75  field ( readonly ) TravelUUID, BookingId;
76  field ( readonly ) CreatedBy, LastChangedBy, LocalLastChangedAt;
77}
```

[TRL] ZI\_RAP\_TRAVEL\_TM3

quarta-feira, 3 de março de 2021, 17:54

```
78 determination calculateBookingID on modify { create; }
79 determination calculateTotalPrice on modify { field FlightPrice, CurrencyCode; }
80
81 mapping for zrap_abook_tm3
82 {
83     BookingUUID      = booking_uuid;
84     TravelUUID       = travel_uuid;
85     BookingID        = booking_id;
86     BookingDate      = booking_date;
87     CustomerID       = customer_id;
88     CarrierID        = carrier_id;
89     ConnectionID     = connection_id;
90     FlightDate       = flight_date;
91     FlightPrice      = flight_price;
92     CurrencyCode     = currency_code;
93     CreatedBy        = created_by;
94     LastChangedBy    = last_changed_by;
95     LocalLastChangedAt = local_last_changed_at;
96 }
97 }
```

Teresa Monteiro, 52597, MIEI FCT

02/03/2021

## Glossário

**Service Instances<sub>1</sub>** - No contexto do SAP Cloud, é a definição de um cliente OAuth.

**Transport request<sub>2</sub>** - é uma espécie de *Container / Collection<sub>3</sub>* de mudanças que são feitas no sistema de desenvolvimento.

**Container / Collection<sub>3</sub>** - Um Container SAP é um controle que acomoda outros controles. Ele gere esses controles de forma lógica numa coleção e fornece uma área física na qual eles são exibidos.

**Core Data Service<sub>4</sub>** (CDS) – Tem o objetivo de suportar o esboço das funcionalidades, da extensibilidade e as implementações “fora da caixa” para assumir todas as implementações técnicas das tarefas.

**Business Object<sub>5</sub>** (BO) - É um termo comum para representar um artefato do mundo da vida real numa aplicação. No caso do meu estágio serão as viagens e as reservas os BO.

**Query<sub>6</sub>** - É a interface de conexão para o acesso somente de leitura à base de dados nos serviços Odata, sendo usado basicamente para relatórios de lista ou relatórios analíticos para processar os dados;

**Business service<sub>7</sub>** - É o conjunto do *service definition<sub>8</sub>* e do *service binding<sub>9</sub>*.

**Service definition<sub>8</sub>** - Descreve quais as entidades do core data service de um modelo de dados que vão ser expostas para um business service específico.

**Service binding<sub>9</sub>** - É um objeto do repositório ABAP usado para ligar um service definition com o protocolo de comunicação cliente-servidor, como *Odata<sub>10</sub>*.

**Odata<sub>10</sub>** - É um protocolo que oferece muitas funcionalidades, como por exemplo, powerful querying, e com a SAP Fiori e UI5, podendo criar Fiori applications.

**Web Api<sub>11</sub>** - É um serviço OData que é exposto como uma API da Web que vem sem nenhuma informação específica da user interface (UI) nos metadados, isto é, a Web api é a interface pública para que qualquer cliente OData possa ter acesso ao serviço OData. Na SAP fiori UI service, a cada configuração front-end que se manifesta no objeto de desenvolvimento de back-end, são expostos nos metadados do serviço. Isso significa que uma UI Fiori lê as informações nos metadados e cria a interface do utilizador correspondente para o serviço. Essas configurações da UI podem ser ampliadas e sobrescritas no SAP Web IDE.

**Sap Fiori<sub>11</sub>** - É uma linguagem de design e abordagem de experiência do utilizador desenvolvida pela SAP para ser usada pela SAP, pelos seus clientes e parceiros em aplicações.

**Namespace<sub>12</sub>** – É o espaço de nomes que pode ser usado para objetos personalizados. Existe para diferenciar entre os padrões SAP e os que são desenvolvidos por o programador.

**Database Migration Option<sub>13</sub>** (/DMO/) – Simplifica e agiliza muito toda a migração SAP com um processo, uma ferramenta e um tempo de inatividade. O DMO permite que os utilizadores SAP atualizem um sistema SAP existente para uma versão superior e/ou podem migrar para a base de dados SAP HANA.

**IF\_OO\_ADT\_RUN<sub>14</sub>** – Na execução da aplicação no eclipse, a classe que implementa if\_oo\_adt\_classrun envia como um parâmetro para o serviço "ADT" como uma solicitação HTTP.

**View<sub>15</sub>** - É criada combinando os dados de uma ou mais tabelas contendo informações sobre um objeto de uma aplicação e pode-se representar um subconjunto dos dados contidos numa tabela ou pode até juntar várias tabelas numa única tabela virtual.

**Data Definition<sub>16</sub>** – Define estruturas.

**Projection view<sub>17</sub>** - Fornece meios dentro de um serviço específico para definir projeções específicas do serviço.

**Alias<sub>18</sub>** - São nomes que os administradores do sistema definem para cada sistema que criam para que os componentes do portal possam fazer referência a esses sistemas. São um meio para recuperar as informações armazenadas em servidores de base de dados sem precisar saber o nome do servidor.

**Semantics annotations<sub>19</sub>** - Permitem a padronização da semântica que só tem impacto no lado do utilizador.

**Value help<sub>20</sub>** – Funcionalidade que permite encontrar a informação (dados) em falta.

**Implementation managed<sub>21</sub>** - Está associada a uma greenfield application, definida anteriormente, que tem o runtime managed.

**Lock master<sub>22</sub>** - O bloqueio é ativado quando o registo mestre é salvo.

**Etag<sub>23</sub>** - É usado para a função do processador, que pode ser modificada. Ao usar uma ETag mestre em todas as entidades, o processamento simultâneo é habilitado para o BO de viagem.

**Key service<sub>24</sub>** - é usada para conectar uma *instance service* à ABAP Development Tools (ADT).

**Abap trial<sub>25</sub>** – é uma parte da Cloud Foundry trial que pode ser acedido através do seguinte link <https://cockpit.hanatrial.ondemand.com>.

**ABAP Class<sub>26</sub>** - é a menor unidade de encapsulamento em objetos ABAP. Um método pode, portanto, usar todas as componentes de todas as instâncias da mesma classe, exceto as componentes da sua própria instância. Uma exceção a essa regra são as subclasses que não podem ter acesso aos componentes privados das superclasses, se não forem seus amigos.

**Dicionário ABAP<sub>27</sub>** - Permite centralizar a definição e a gestão de tipos como por exemplo, a criação de elementos de dados definidos pelo utilizador, estruturas e tipos de tabela, a definição de tabelas, índices e visualizações e a definição dos serviços que suportam o desenvolvimento do programa.

**PFCG<sub>28</sub>** – É um código transaccional que administra a manutenção de funções para gerir funções e dados de autorização.

**Savepoint<sub>26</sub>** - <https://docs.oracle.com/en/database/oracle/oracle-database/19/jjdbc/JDBC-standards-support.html#GUID-020AF373-6BF0-4BB3-B338-6F9609A223CA>

**Field level help<sub>27</sub>** - [https://docs.oracle.com/cd/E59116\\_01/doc.94/e58804/ch\\_locate\\_field\\_help.htm#WEABA219](https://docs.oracle.com/cd/E59116_01/doc.94/e58804/ch_locate_field_help.htm#WEABA219)  
[https://docs.oracle.com/cd/E59116\\_01/doc.94/e58804/ch\\_locate\\_prog Lev\\_help.htm#WEABA203](https://docs.oracle.com/cd/E59116_01/doc.94/e58804/ch_locate_prog Lev_help.htm#WEABA203)



## Referências:

Database System Concepts, Seventh Edition, Abraham Silberschatz, Henry F. Korth, S.Sudar shan, LCCN 2018060474 | ISBN 9780078022159 (alk. paper) | ISBN 0078022150

<https://apex.oracle.com/en/learn/documentation/> (02/03/2021)

<https://open.sap.com/courses/cp13/overview> (02/03/2021)

<https://accenture.percipio.com/books/264e4dd8-1778-493d-81d7-a36b376ece9a> (02/03/2021)

[https://help.sap.com/doc/3750bcdf7b8045e18f1b759e6d2b000b/Cloud/en-US/ABAP\\_RESTful\\_Programming\\_Model\\_EN.pdf](https://help.sap.com/doc/3750bcdf7b8045e18f1b759e6d2b000b/Cloud/en-US/ABAP_RESTful_Programming_Model_EN.pdf) (02/03/2021)

<https://blogs.sap.com/2019/02/08/evolution-of-the-abap-programming-model/#ClassicABAPProgramming> (02/03/2021)

<https://github.com/SAP-samples/abap-platform-rap-opensap> (02/03/2021)

<https://www.sapinsideronline.com/articles/a-developers-guide-to-the-abap-restful-programming-model/> (02/03/2021)

<https://blogs.sap.com/2019/02/08/evolution-of-the-abap-programming-model/> (02/03/2021)

<https://blogs.sap.com/2020/09/21/comparing-abap-restful-application-programming-rap-model-with-the-cloud-application-programming-cap-model/> (02/03/2021)

<https://www.sapinsideronline.com/articles/a-developers-guide-to-the-abap-restful-programming-model/> (02/03/2021)

<https://blogs.sap.com/2016/04/04/getting-started-abap-programming-model/> (02/03/2021)

<https://help.sap.com/viewer/index> (02/02/2021)

Teresa Monteiro, 52597, MIEI FCT

02/03/2021