

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

```

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 }

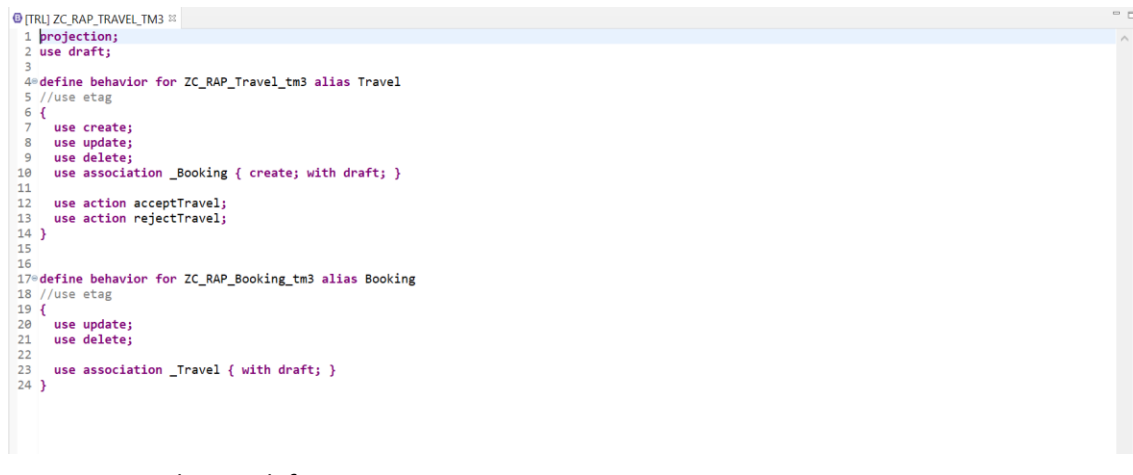
```

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*₂₃ 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.

```

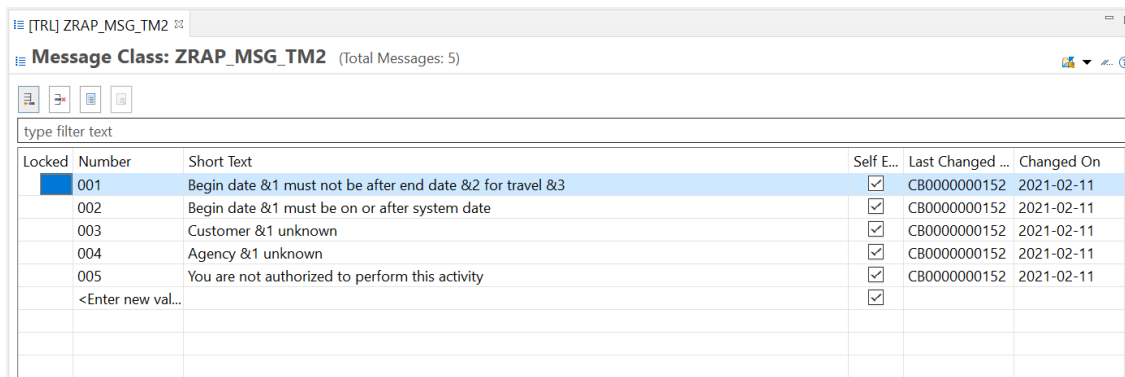
[TRL] ZCL_RAP_EML_TM3
1 @CLASS zcl_rap_eml_tm3
2 PUBLIC
3 FINAL
4 CREATE PUBLIC .
5
6 PUBLIC SECTION.
7
8 INTERFACES if_oo_adt_classrun.
9 PROTECTED SECTION.
10 PRIVATE SECTION.
11 ENDCLASS.
12
13
14
15 @CLASS zcl_rap_eml_tm3 IMPLEMENTATION.
16 @METHOD if_oo_adt_classrun~main.
17 * " step 1 - READ
18 * READ ENTITIES OF ZI_RAP_Travel_tm3
19 * ENTITY travel
20 * FROM VALUE #( ( TravelUUID = '1D921CB0180A116517000A0274216DF0' ) )
21 * RESULT DATA(travels).
22 *
23 * out->write( travels ).
24
25 * " step 2 - READ with Fields
26 * READ ENTITIES OF ZI_RAP_Travel_tm3
27 * ENTITY travel
28 * FIELDS ( AgencyID CustomerID )
29 * WITH VALUE #( ( TravelUUID = '1D921CB0180A116517000A0274216DF0' ) )
30 * RESULT DATA(travels).
31 *
32 * out->write( travels ).
33
34 * " step 3 - READ with All Fields
35 * READ ENTITIES OF ZI_RAP_Travel_tm3
36 * ENTITY travel
37 * ALL FIELDS
38 * WITH VALUE #( ( TravelUUID = '1D921CB0180A116517000A0274216DF0' ) )
39 * RESULT DATA(travels).
40 *
41 * out->write( travels ).
42
43 * " step 4 - READ By Association
44 * READ ENTITIES OF ZI_RAP_Travel_tm3
45 * ENTITY travel BY \Booking
46 * ALL FIELDS WITH VALUE #( ( TravelUUID = '1D921CB0180A116517000A0274216DF0' ) )
47 * RESULT DATA(bookings).
48 *
49 * out->write( bookings ).
50
51 * " step 5 - Unsuccessful READ
52 * READ ENTITIES OF ZI_RAP_Travel_tm3
53 * ENTITY travel
54 * ALL FIELDS WITH VALUE #( ( TravelUUID = '11111111111111111111111111111111' ) )
55 * RESULT DATA(travels)
56 * FAILED DATA(failed)
57 * REPORTED DATA(reported).
58 *
59 * out->write( travels ).
60 * out->write( failed ). " complex structures not supported by the console output
61 * out->write( reported ). " complex structures not supported by the console output
62
63 * " step 6 - MODIFY Update
64 * MODIFY ENTITIES OF ZI_RAP_Travel_tm3
65 * ENTITY travel
66 * UPDATE
67 * SET FIELDS WITH VALUE
68 * #( ( TravelUUID = '1D921CB0180A116517000A0274216DF0'
69 * Description = 'I like RAP@openSAP' ) )
70
71 * FAILED DATA(failed)
72 * REPORTED DATA(reported).
73 *
74 * " step 6b - Commit Entities
75 * COMMIT ENTITIES
76 * RESPONSE OF ZI_RAP_Travel_tm3
77 * FAILED DATA(failed_commit)
78 * REPORTED DATA(reported_commit).
79 *
80 * out->write( 'Update done' ).
81
82 * " step 7 - MODIFY Create
83 * MODIFY ENTITIES OF ZI_RAP_Travel_tm3
84 * ENTITY travel
85 * CREATE
86 * SET FIELDS WITH VALUE
87 * #( ( %cid = 'MyContentID_1'
88 * AgencyID = '70012'
89 * CustomerID = '14'
90 * BeginDate = c1_abap_context_info=>get_system_date( )
91 * EndDate = c1_abap_context_info=>get_system_date( ) + 10
92 * Description = 'I like RAP@openSAP' ) )
93 *
94 * MAPPED DATA(mapped)
95 * FAILED DATA(failed)
96 * REPORTED DATA(reported).
97 *
98 * out->write( mapped-travel ).
99

```

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
--

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;
--
```

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*²⁷ 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*²⁷ 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*²⁹. 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*²⁶ 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.