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 constroí, 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*$_{1}$ criada do tipo *abap trial*$_{25}$ (figura 1), e dentro desta criamos uma nova instância (figura 2). Dentro desta instância criamos uma *key service*$_{24}$. 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 perspetiva 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).

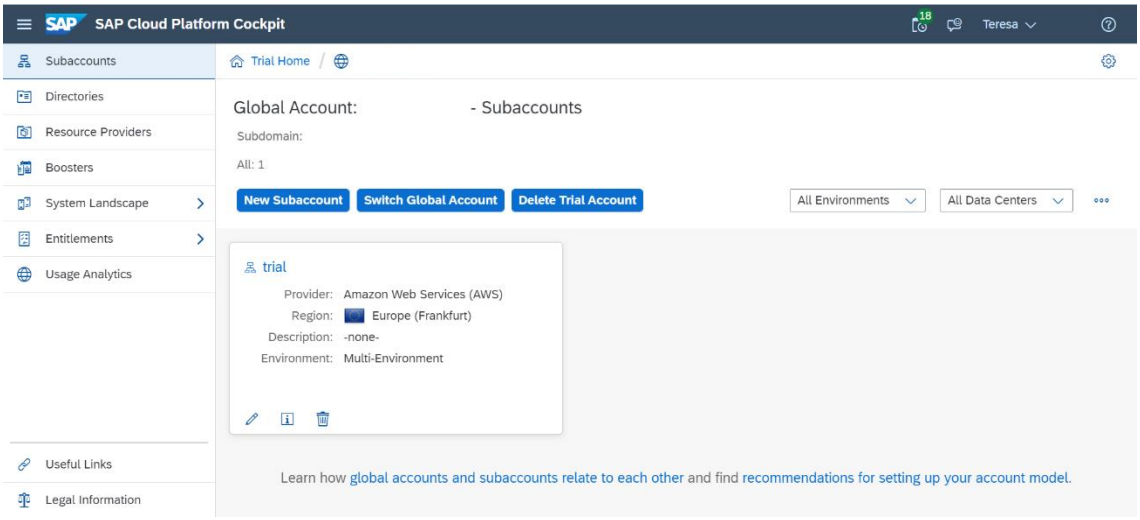


Figura 1: Ambiente SAP com conta trial

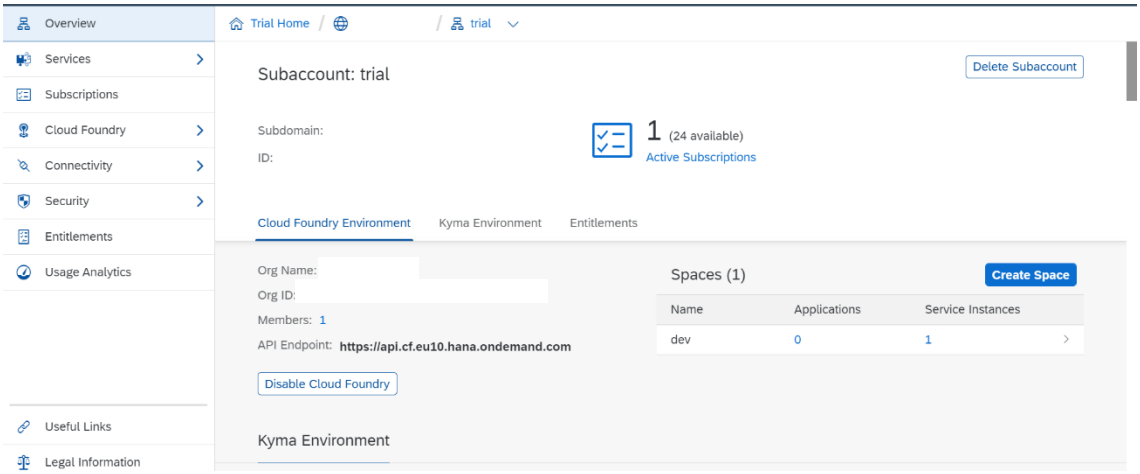


Figura 2: Conta trial com a service instances que se a selecionarmos, vamos poder criar a key.

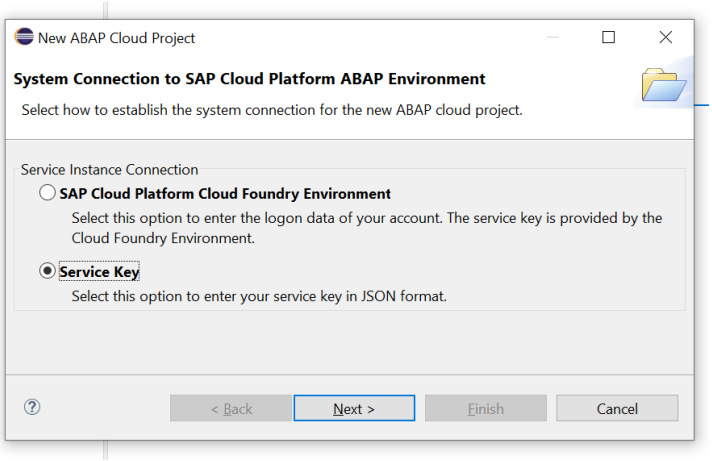


Figura 3: Opções de como é feita a conexão entre o eclipse e o site SAP.

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

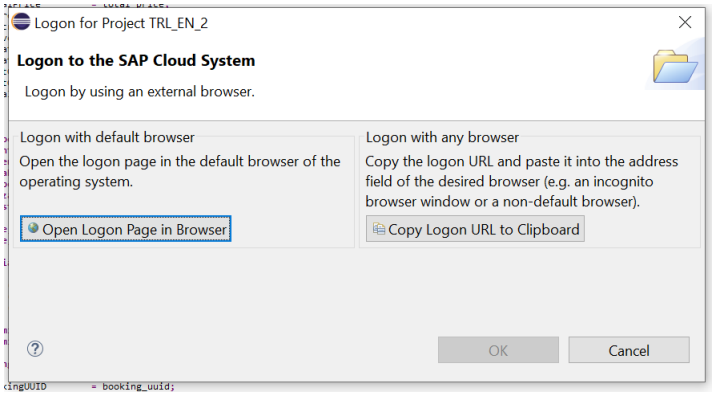
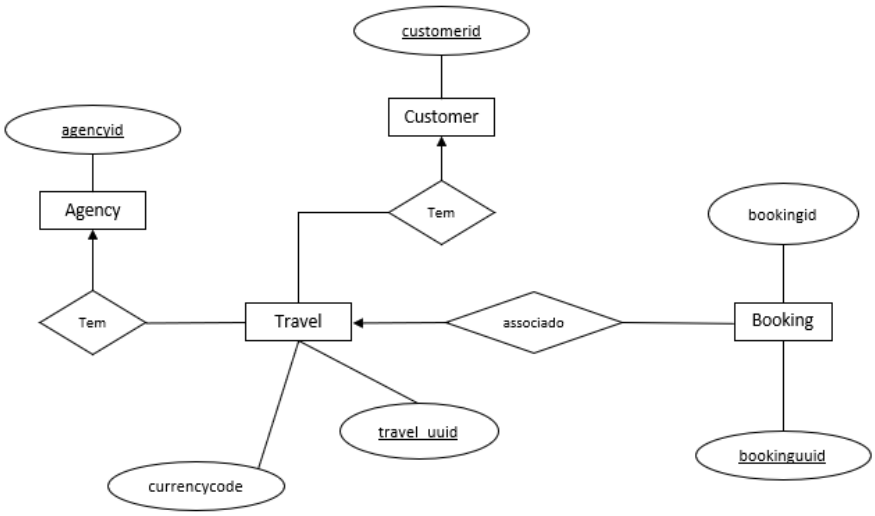


Figura 4: Logon to the SAP Cloud System. Este logon é necessário porque apesar de estarmos a desenvolver no eclipse, fica tudo em cloud.

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 entres 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*[2] *(figura* 5), que também terá que ser criado. Este transport request vai estar sempre associado a qualquer funcionalidade que criamos.

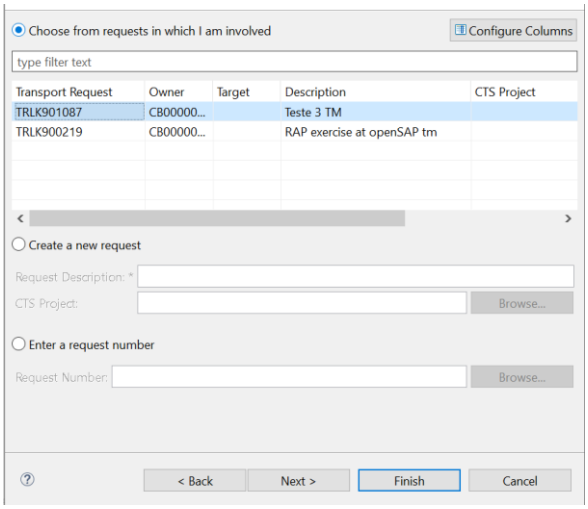


Figura 5: Transport Request

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*[4], a projeção do modelo de dados e o enriquecimento com a semântica da interfaco do utilizador. Em seguida, criaremos o serviço *Odata*[10] e visualizaremos a aplicação de viagens *SAP Fiori*[11]. 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 vôo ABAP pertence ao *namespace*[12] */ DMO /*[13], 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.

Comecemos 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).

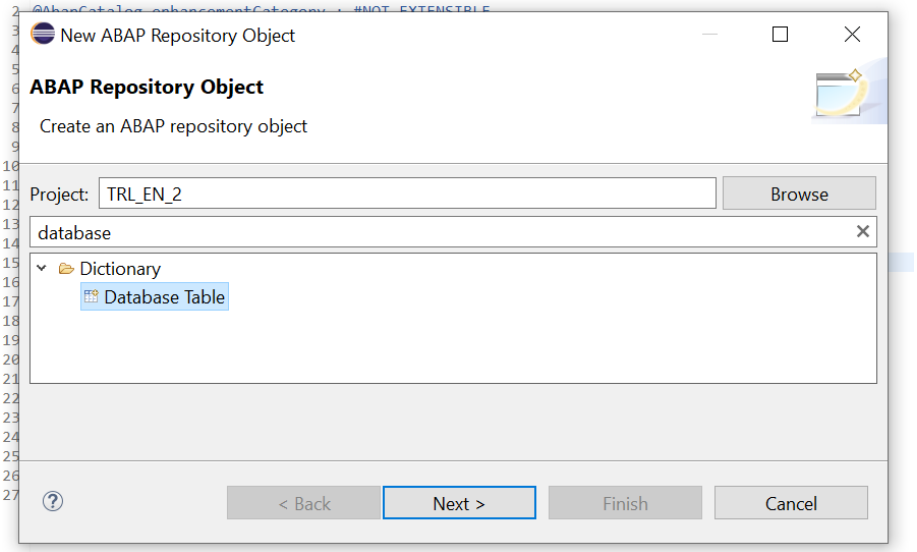


Figura 7: Mostra a criação da estrutura database.

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*[26]. A anotação *@AbapCatalog* define as configurações técnicas de entidades CDS no *Dicionário ABAP*[27], existem vários tipos de anotações relacionados (figura 8).

[TRL] ZRAP_ATRAV_TM3

```
@EndUserText.label : 'Travel data'
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table zrap_atrav_tm3 {
  key client           : mandt not null;
  key travel_uuid      : sysuuid_x16 not null;
  travel_id            : /dmo/travel_id;
  agency_id            : /dmo/agency_id;
  customer_id          : /dmo/customer_id;
  begin_date           : /dmo/begin_date;
  end_date             : /dmo/end_date;
  @Semantics.amount.currencyCode : 'zrap_atrav_tm3.currency_code'
  booking_fee          : /dmo/booking_fee;
  @Semantics.amount.currencyCode : 'zrap_atrav_tm3.currency_code'
  total_price          : /dmo/total_price;
  currency_code        : /dmo/currency_code;
  description          : /dmo/description;
  overall_status       : /dmo/overall_status;
  created_by           : syuname;
  created_at           : timestampl;
  last_changed_by      : syuname;
  last_changed_at      : timestampl;
  local_last_changed_at : timestampl;

}
```

[TRL] ZRAP_ABOOK_TM3

```
@EndUserText.label : 'Booking data'
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table zrap_abook_tm3 {
  key client           : mandt not null;
  key booking_uuid     : sysuuid_x16 not null;
  travel_uuid          : sysuuid_x16 not null;
  booking_id           : /dmo/booking_id;
  booking_date         : /dmo/booking_date;
  customer_id          : /dmo/customer_id;
  carrier_id           : /dmo/carrier_id;
  connection_id        : /dmo/connection_id;
  flight_date          : /dmo/flight_date;
  @Semantics.amount.currencyCode : 'zrap_abook_tm3.currency_code'
  flight_price         : /dmo/flight_price;
  currency_code        : /dmo/currency_code;
  created_by           : syuname;
  last_changed_by      : syuname;
  local_last_changed_at : timestampl;

}
```

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*[14]*.* 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.

[TRL] ZCL_GENERATE_DEMO_DATA_TM3

```
CLASS zcl_generate_demo_data_tm3 DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
    INTERFACES if_oo_adt_classrun.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

```

[TRL] ZCL_GENERATE_DEMO_DATA_TM3

```
CLASS zcl_generate_demo_data_tm3 IMPLEMENTATION.
  METHOD if_oo_adt_classrun~main.

    " delete existing entries in the database table
    DELETE FROM zrap_atrav_tm3.
    DELETE FROM zrap_abook_tm3.

    " insert travel demo data
    INSERT zrap_atrav_tm3 FROM (
        SELECT
          FROM /dmo/travel
          FIELDS
            uuid( )       AS travel_uuid           ,
            travel_id     AS travel_id             ,
            agency_id     AS agency_id             ,
            customer_id   AS customer_id           ,
            begin_date    AS begin_date            ,
            end_date      AS end_date              ,
            booking_fee   AS booking_fee           ,
            total_price   AS total_price           ,
            currency_code AS currency_code         ,
            description   AS description           ,
            CASE status
              WHEN 'B' THEN 'A' " accepted
              WHEN 'X' THEN 'X' " cancelled
              ELSE 'O'          " open
            END           AS overall_status        ,
            createdby     AS created_by            ,
            createdat     AS created_at            ,
            lastchangedby AS last_changed_by       ,
            lastchangedat AS last_changed_at       ,
            lastchangedat AS local_last_changed_at
            ORDER BY travel_id UP TO 200 ROWS
      ).
    COMMIT WORK.

    " insert booking demo data
    INSERT zrap_abook_tm3 FROM (
        SELECT
          FROM   /dmo/booking    AS booking
            JOIN zrap_atrav_tm3 AS z
            ON   booking~travel_id = z~travel_id
          FIELDS
            uuid( )                AS booking_uuid          ,
            z~travel_uuid          AS travel_uuid           ,
            booking~booking_id     AS booking_id            ,
            booking~booking_date   AS booking_date          ,
            booking~customer_id    AS customer_id           ,
            booking~carrier_id     AS carrier_id            ,
            booking~connection_id  AS connection_id         ,
            booking~flight_date    AS flight_date           ,
            booking~flight_price   AS flight_price          ,
            booking~currency_code  AS currency_code         ,
            z~created_by           AS created_by            ,
            z~last_changed_by      AS last_changed_by       ,
            z~last_changed_at      AS local_last_changed_by
      ).
    COMMIT WORK.

    out->write( 'Travel and booking demo data inserted.').
  ENDMETHOD.

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*$_5$ *(BO)*. Assim, começamos com a criação da *view*$_{15}$ da viagem: clica-se na tabela da viagem e seleciona-se New *Data Definition*$_{16}$. A tabela das viagens é usada como fonte de dados e os campos da tabela são inseridos automaticamente na lista das projeções entre as chaves. A seguir, definimos os *aliases*$_{18}$ 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*[17]. Uma associação é definida com uma cardinalidade, uma entidade CDS de destino, um *alias*[18] opcional e uma condição on. Na lista de projeções, o elemento de visualização `overall_status` é renomeado para *TravelStatus*.
Depois, podemos adicionar *semantics annotations*[19] 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 *transacional 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 object*[5]. 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).

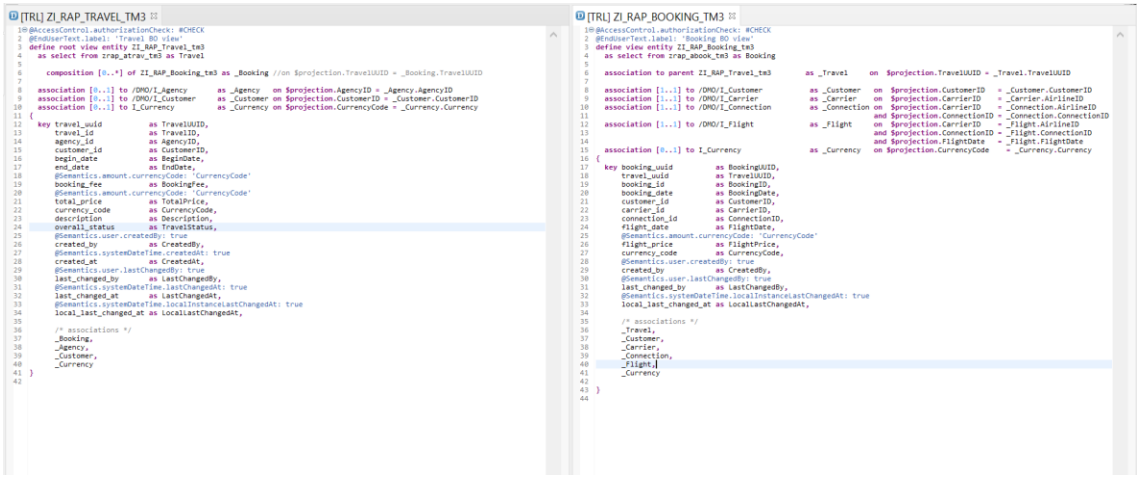


Figura 10: Data definition.

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 especifíca o element view para ser considerado numa pesquisa. A linha 4 está a permitir que haja facilidade nas modificações de ambas as *projections views*[17] 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