Converged Database: Modern Development Workshop
Day 3

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

Índice

REQUERIMIENTOS INICIALES Y ACCESO AL ENTORNO **3**

SESIÓN 3 ..... **6**

DESCRIPCIÓN DEL ENTORNO DEL WORKSHOP **6**
OPTIMIZADOR DE RUTAS (TRAVELING SALESPERSON PROBLEM) **6**
PETICIÓN TSP 7
APLICACIÓN APEX PARA VISUALIZAR LAS RUTAS 10
CREACIÓN DE UN RECOMENDADOR USANDO *PROPERTY GRAPH* O GRAFO. **22**
CREACIÓN DE VISTAS EN APP_ROOT 23
CREACIÓN DE GRAFO EN APP_ROOT 25
ACCESO AL GRAFO DESDE JAVA 27
MODELO PREDICTIVO PYTHON IN-DATABASE Y API REST **30**
CREACIÓN DEL MODELO PREDICTIVO 30
EJECUCIÓN EMBEBIDA DEL MODELO PREDICTIVO DESDE PYTHON 35
EJECUCIÓN DEL MODELO PREDICTIVO DESDE SQL 36
EXPOSICIÓN Y EJECUCIÓN DEL MODELO PREDICTIVO DESDE REST 37

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

Requerimientos iniciales y acceso al entorno

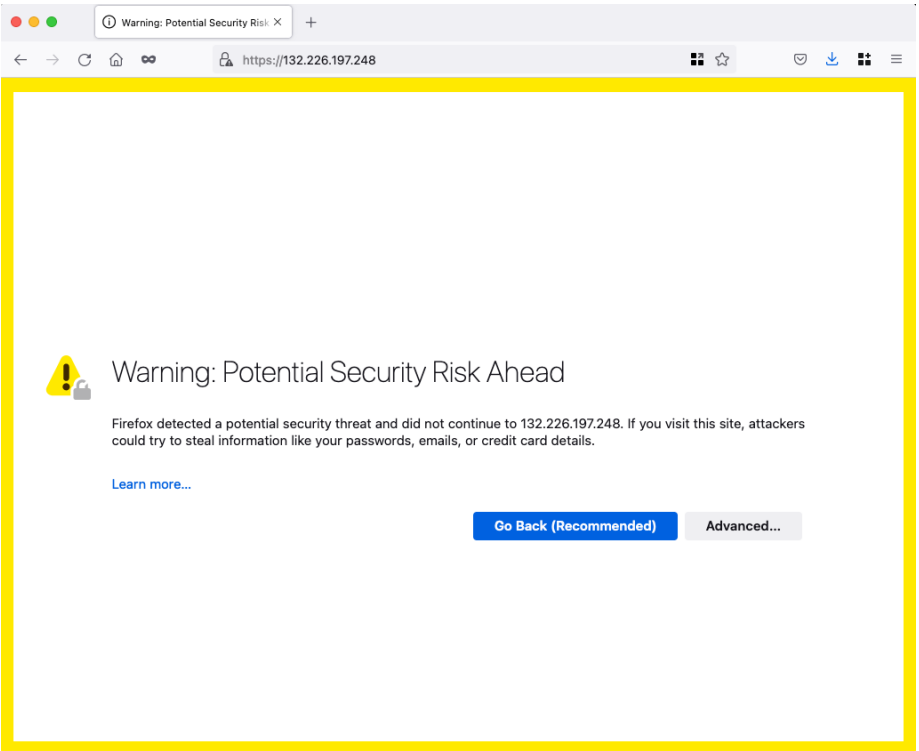
El taller que va a realizar está diseñado para realizarse usando únicamente un navegador web desde el que podrá tener acceso a un escritorio remoto.

Para la realización de este taller necesita:

- Clave de acceso al entorno que le proporcionará el instructor. Debe acceder con el usuario **oracle**.
- Navegador web (Firefox o Chrome recomendados)
- Dirección IP de la maquina.

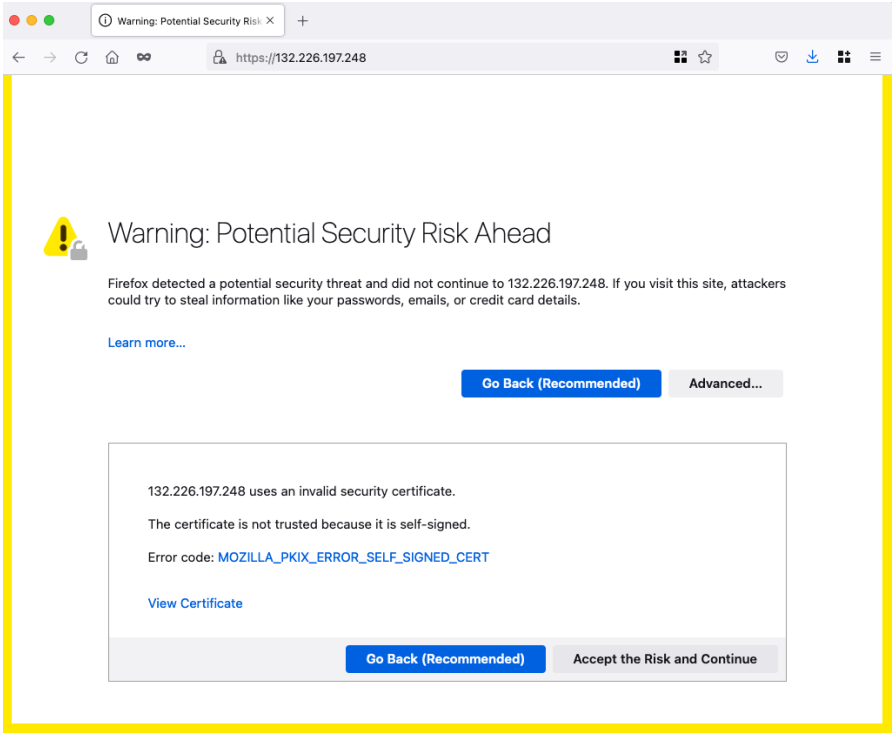
La conexión se realiza por https, pero el certificado es autogenerado por lo que recibirá una alerta de seguridad en el primer intento de conexión informando de este hecho. Debe aceptar el mensaje para poder continuar con la conexión.

Haga click en el botón 'Advanced…':

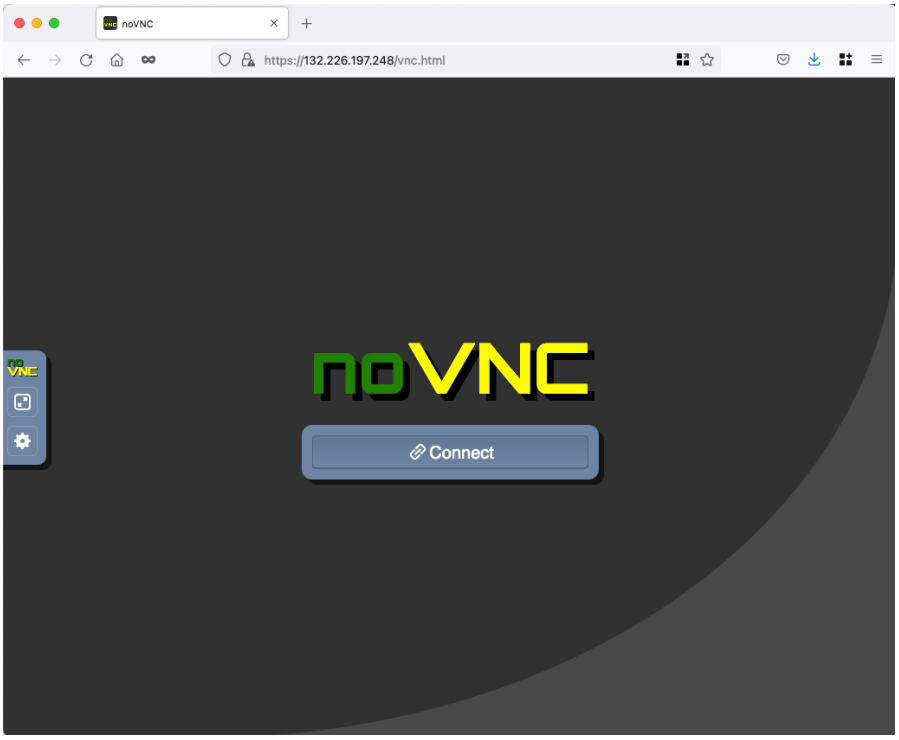


Verá el detalle del potencial problema de seguridad, que es debido al uso de un certificado autogenerado.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

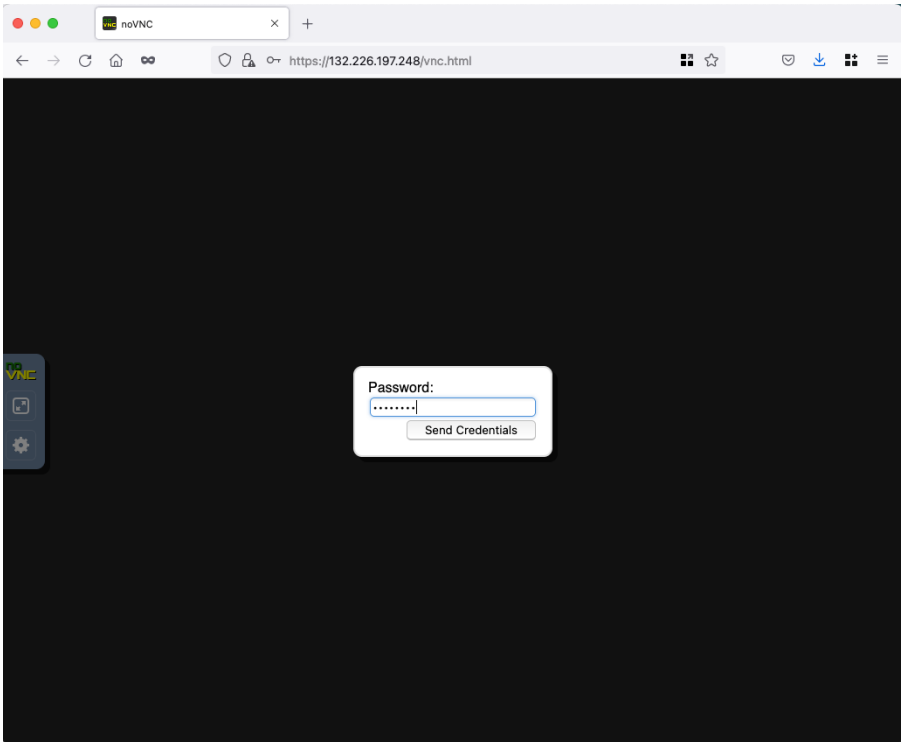


Pulse el botón ‘Acept the Risk and Continue’ tras lo cual deberá ver la pantalla de acceso a noVNC, que proporciona el acceso al escritorio remoto.

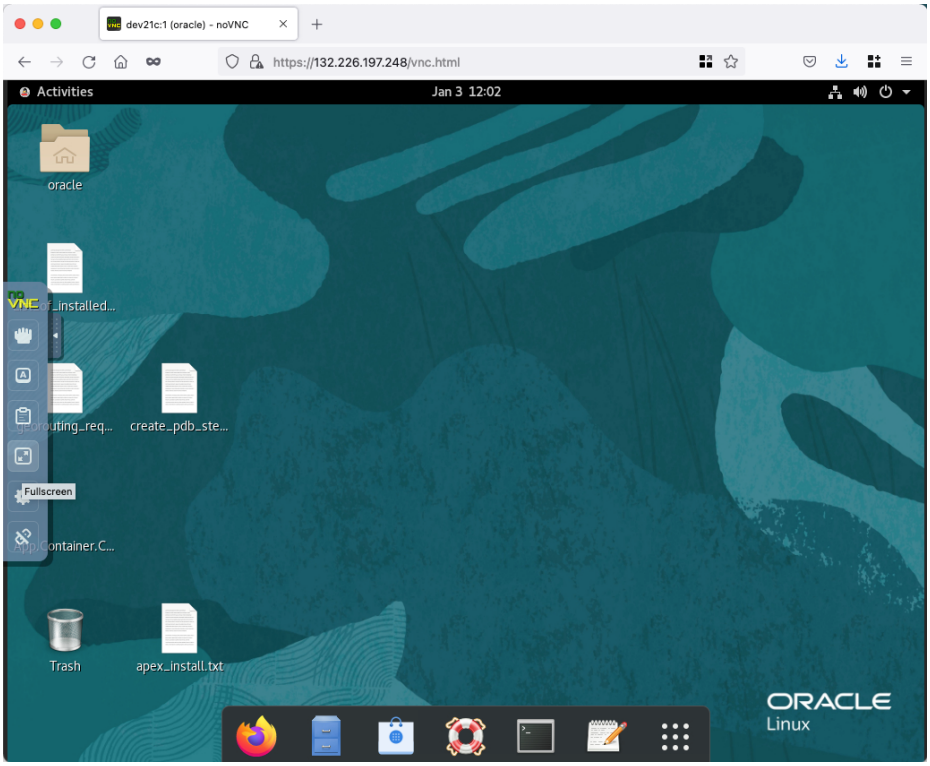


Haga click en el botón ‘Connect’ e introduzca la clave proporcionada por el instructor.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.



Tras estos pasos verá en su pantalla el escritorio remoto de la máquina del workshop. En el menú del lateral izquierdo puede hacer que el escritorio ocupe toda la pantalla (opción Fullscreen).

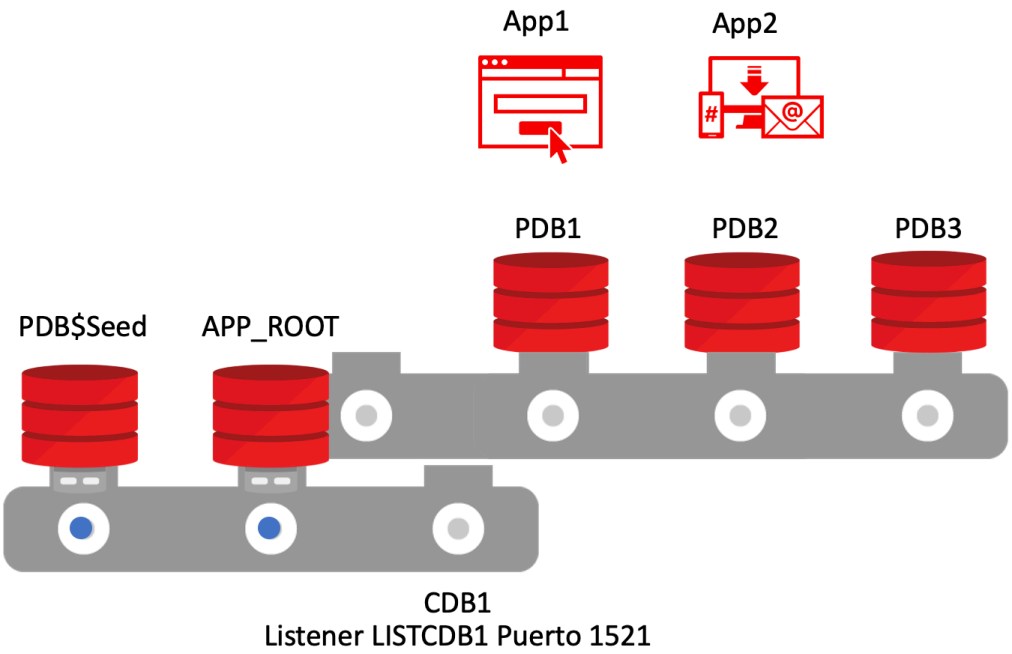


Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

Sesión 3

Descripción del entorno del workshop

La máquina debe tener un estado como se muestra en este diagrama después de las actividades de los días 1 y 2 de este taller.



Recuerde que puede usar SQL*Developer para interactuar con las distintas PDB y esquemas. En cada PDB existe un esquema de trabajo llamado soe con clave soe.

Optimizador de rutas (Traveling Salesperson Problem)

El problema del viajante (Traveling Salesperson Problem) está contemplado en el motor de cálculo de rutas de Oracle. Se trata de partiendo de un punto, recorrer varias direcciones de forma óptima pudiendo volver al punto de salida (ruta cerrada) o bien teniendo un punto de llegada diferente al de salida (ruta abierta).

En esta actividad se va a usar una clase Java que hace una solicitud de ruta cerrada con tres puntos intermedios a visitar al motor de rutas de Oracle. La respuesta se va a encapsular en formato

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

GeoJSON (RFC 7946) y posteriormente se va a visualizar a través de una aplicación APEX. El formato GeoJSON está ampliamente soportado por herramientas y lenguajes de programación.

Con esta funcionalidad se puede optimizar el recorrido de los repartidores, ayudando en la planificación de las rutas y los recorridos.

Petición TSP

El código de la aplicación está en el repositorio github usado anteriormente.
Si necesita volver a clonarlo desde un terminal, ejecute este comando:

```
git clone https://github.com/OracleDataManagementSpain/ConvergedDatabase/
```

Cambie al directorio ConvergedDatabase/Developers, descomprima el fichero java_libraries.zip

```
❯ git clone https://github.com/OracleDataManagementSpain/ConvergedDatabase/
Cloning into 'ConvergedDatabase'...
remote: Enumerating objects: 694, done.
remote: Counting objects: 100% (185/185), done.
remote: Compressing objects: 100% (149/149), done.
remote: Total 694 (delta 87), reused 64 (delta 33), pack-reused 509
Receiving objects: 100% (694/694), 448.66 MiB | 27.08 MiB/s, done.
Resolving deltas: 100% (301/301), done.
❯ cd ConvergedDatabase/Developers
❯ unzip java_libraries.zip
Archive:  java_libraries.zip
  inflating: okhttp-3.9.1.jar
  inflating: javax.json-1.1.4.jar
  inflating: json-simple-1.1.1.jar
  inflating: okio-1.13.0.jar
```

La clase que implementa la petición TSP al motor de rutas es TSPRequestDBtoGeoJson.java.
Contiene un único método:

```
public static Clob getGeoJson(String start_loc, String loc_2,String loc_3)
```

Que espera recibir la dirección inicial y dos puntos intermedios a visitar, siendo la ruta cerrada.
Las direcciones se pasarán en formato:

```
street;city;postalcode;region
```

Siendo el punto y coma (;) el separador entre ellas.

La respuesta del motor de rutas se parsea y con la librería json-simple se construye siguiendo la especificación GeoJSON (RFC 7946) un documento JSON con la estructura necesaria para contener la información de la ruta así como algunas otras propiedades (distancia, tiempo). Este documento se va a almacenar en una columna JSON en una tabla de la base de datos.

El objeto resultante será un CLOB ya que la geometría de la ruta puede contener un número alto de posiciones geolocalizadas dentro del GeoJSON. Para construir el objeto Clob se usa una conexión a la base de datos, que se construye fácilmente:

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
Connection conn = DriverManager.getConnection("jdbc:default:connection:");
```

A través de la cual se crea el Clob a partir del GeoJSON a final del método.

```
String g = JSONValue.toJSONString(geojson);
Clob myClob = conn.createClob();
myClob.setString(1,g);
```

A continuación compile la clase Java con la siguiente sentencia. En este caso es necesario incluir la librería ojdbc11.jar por la conexión que se hace a la base de datos.

```
❯ $ORACLE_HOME/jdk/bin/javac -cp "json-simple-
1.1.1.jar:$ORACLE_HOME/jdbc/lib/ojdbc11.jar" TSPRequestDBtoGeoJson.java
```

La librería ojdbc11.jar usada en la compilación no es necesario cargarla en el esquema de la base de datos, pero la json-simple-1.1.1.jar sí. Esta nueva clase se va a cargar en el esquema SOE de la PDB3.

```
❯ loadjava -user soe/soe@localhost:1521/pdb3 -genmissing -r json-simple-1.1.1.jar
TSPRequestDBtoGeoJson.class
```

El usuario soe ya dispone de algunos permisos necesarios para realizar conexiones a http://maps.oracle.com que se le concedieron en el taller del día 2, pero en esta ocasión se está conectando desde Java por lo que es necesario darle algunos más:

```
❯ sqlplus sys/Oracle_4U@localhost:1521/pdb3 as sysdba

SQL*Plus: Release 21.0.0.0.0 - Production on Tue Jan 11 12:40:43 2022
Version 21.3.0.0.0

Copyright (c) 1982, 2022, Oracle.  All rights reserved.


Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> exec dbms_java.grant_permission('SOE', 'SYS:java.net.SocketPermission',
'maps.oracle.com:80', 'connect,resolve');

PL/SQL procedure successfully completed.

SQL> exit
```

A continuación se crea la función que encapsula la llamada a la función Java que se acaba de cargar.

```
❯ sqlplus soe/soe@localhost/pdb3

SQL*Plus: Release 21.0.0.0.0 - Production on Mon Apr 11 17:42:59 2022
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
Last Successful login time: Mon Apr 11 2022 17:35:29 +00:00

Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> CREATE OR REPLACE FUNCTION tsp_geojson_j (start_loc varchar2, loc_2 varchar2, loc3
varchar2) RETURN CLOB
AS LANGUAGE JAVA
NAME
'TSPRequestDBtoGeoJson.getGeoJson(java.lang.String,java.lang.String,java.lang.String)
return java.sql.Clob';
/

Function created.
```

Se puede realizar una validación del proceso hasta este punto realizando una llamada a la función:

```
SQL> select tsp_geojson_j ('Calle Urca 6;Majadahonda;28220;Madrid','Calle Gabriel Garcia
Marzquez 2;Las Rozas de Madrid;28232;Madrid','Calle Hijuela 6;Villaviciosa de
Odon;28670;Madrid') from dual;

TSP_GEOJSON_J('CALLEURCA6;MAJADAHONDA;28220;MADRID','CALLEGABRIELGARCIAMARZQUEZ2
--------------------------------------------------------------------------------
{"type":"FeatureCollection","features":[{"type":"Feature","properties":{"dist":"
```

Cree una tabla con una columna JSON para poder almacenar esta información:

```
SQL> CREATE TABLE tspgeojson (
  id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY (CACHE 5) PRIMARY KEY,
  geojson json);

Table created.
```

Realice varias peticiones y almacene los resultados en ella:

```
SQL> insert into tspgeojson (geojson) values (tsp_geojson_j ('Calle Jose Echegaray 6;Las
Rozas de Madrid;28232;Madrid','Avenida Mirasierra 1;Villanueva de la
Cañada;28691;Madrid','Calle Hijuela 6;Villaviciosa de Odon;28670;Madrid'));

1 row created.

SQL> insert into tspgeojson (geojson) values (tsp_geojson_j ('Calle Jose Echegaray 6;Las
Rozas de Madrid;28232;Madrid','Calle Picasso 13;Valdemorillo;28210;Madrid','Avenida de
los Toreros 2;Sevilla la Nueva;28609;Madrid'));

1 row created.

SQL> insert into tspgeojson (geojson) values (tsp_geojson_j ('Calle Jose Echegaray 6;Las
Rozas de Madrid;28232;Madrid','Calle Madrid 39;Brunete;28690;Madrid','Camino de las
Huertas 14;Pozuelo de Alarcón;28223;Madrid'));

1 row created.

SQL> commit;
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
Commit complete.

SQL> exit
```

Aplicación APEX para visualizar las rutas

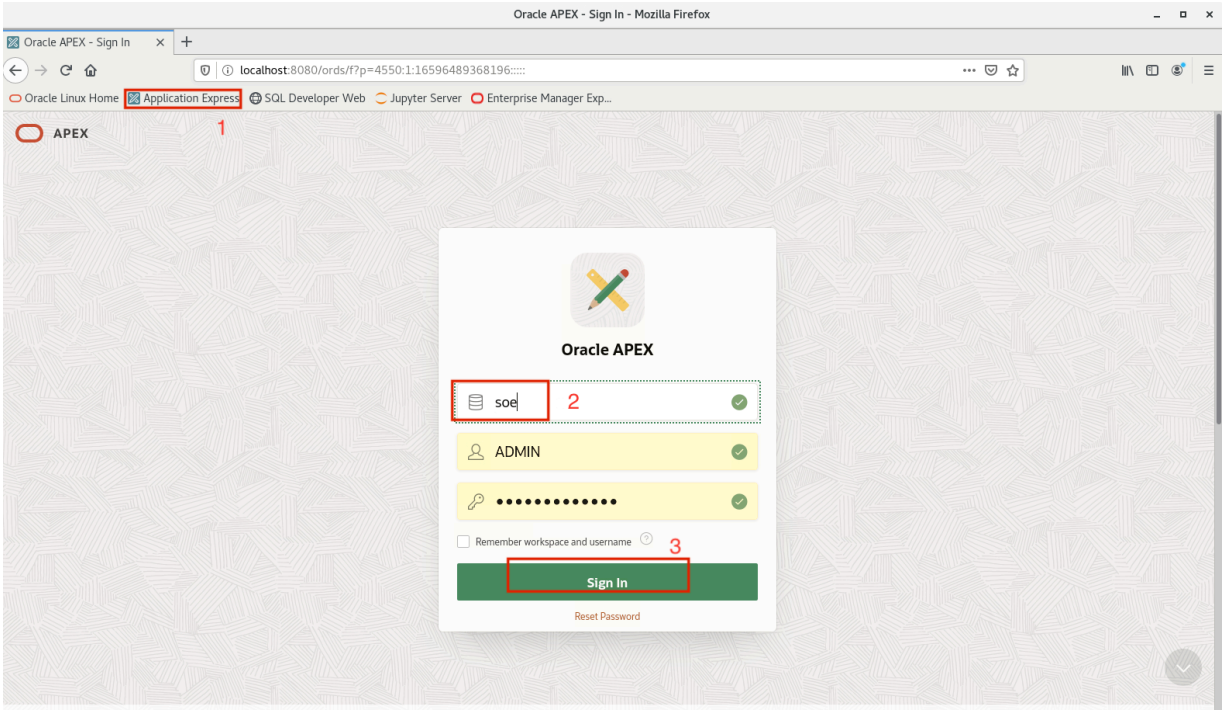
Para visualizar estas rutas, desde APP_ROOT se crea una tabla vacía igual a la de PDB3 que contiene las rutas (`tsggeojson`) y una vista que extraiga del GeoJSON la línea de la ruta así como algunas otras propiedades que e incluyeron como la distancia y el tiempo estimado.

```
❯ sqlplus soe/soe@localhost/app_root

SQL*Plus: Release 21.0.0.0.0 - Production on Mon Apr 11 18:21:31 2022
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.

Last Successful login time: Mon Apr 11 2022 13:39:43 +00:00

Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> CREATE TABLE tspgeojson (
 id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY (CACHE 5) PRIMARY KEY,
 geojson json);

Table created.

SQL> create or replace view tspgeojson_v as
select j.id, jt.dist, jt.duration, jt.sdo_val
from containers(tspgeojson) j NESTED geojson.features[*]
COLUMNS (sdo_val SDO_GEOMETRY PATH '$.geometry',
         dist varchar2 PATH '$.properties.dist',
         duration varchar2 PATH '$.properties.time') jt;

View created.

SQL> exit
```

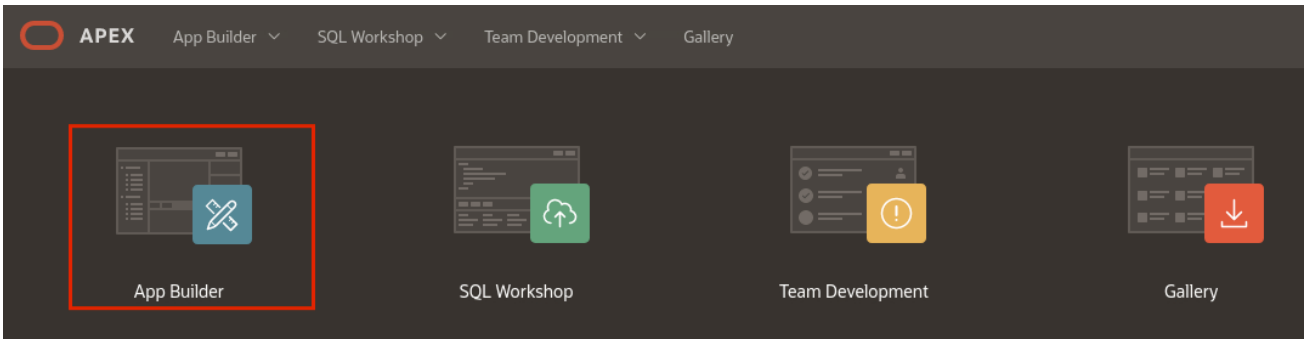
Con la vista `tspgeojson_v` desde APP_ROOT se pueden consultar todas las rutas que se generen en PDB3.

Abra el navegador Firefox, use el marcador **Application Express** de la barra de direcciones (1). En la pantalla de login introduzca el nombre del workspace SOE (2) y haga click en el botón Sign In.

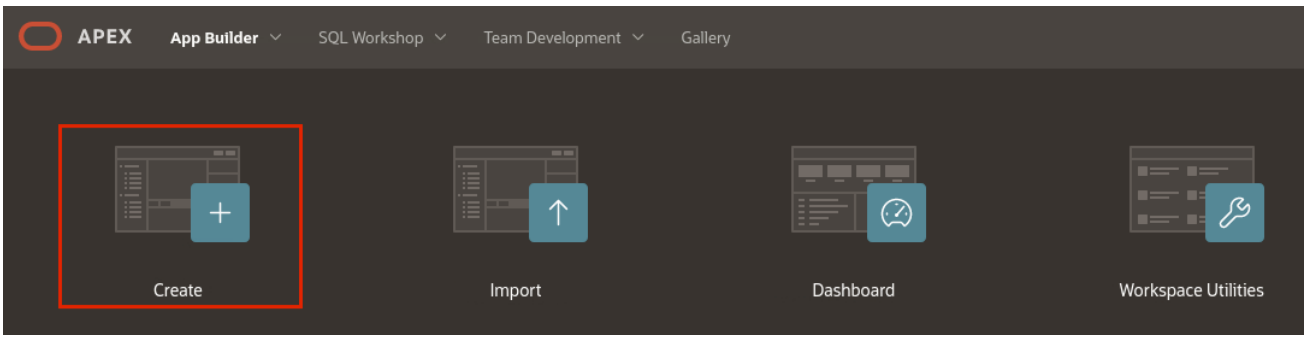
Copyright © 2022, Oracle and/or its affiliates. All rights reserved.



Seleccione App Builder en la zona superior izquierda de la pantalla:

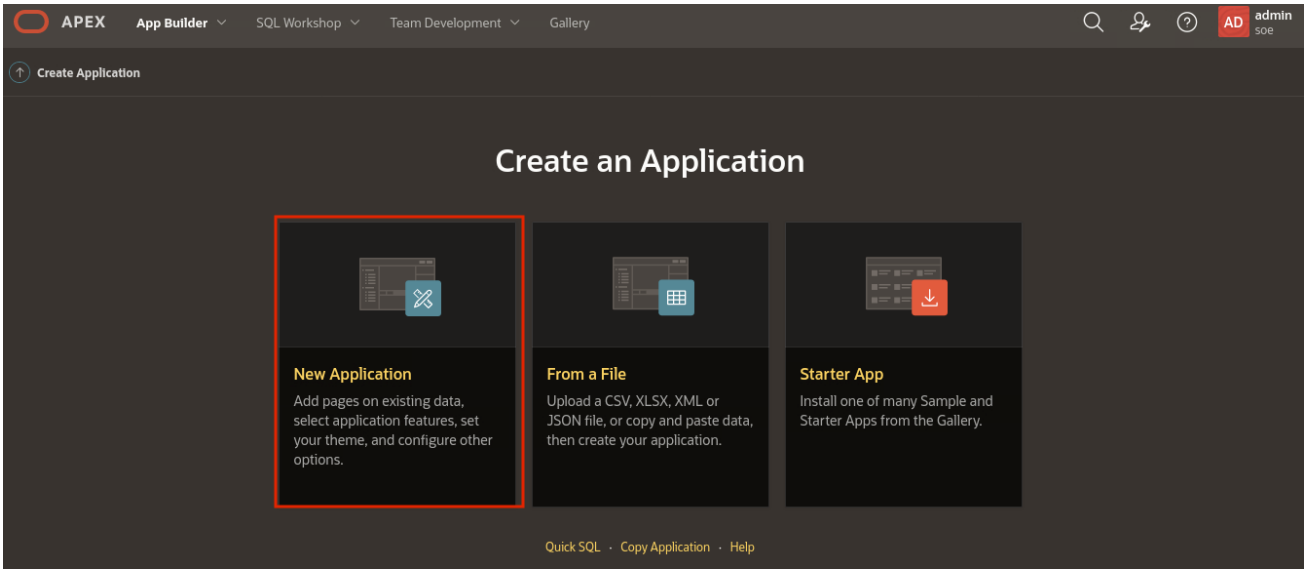


Se mostrarán las herramientas para la creación de aplicaciones APEX. De nuevo, en la zona superior izquierda, escoja Create para construir la nueva aplicación.

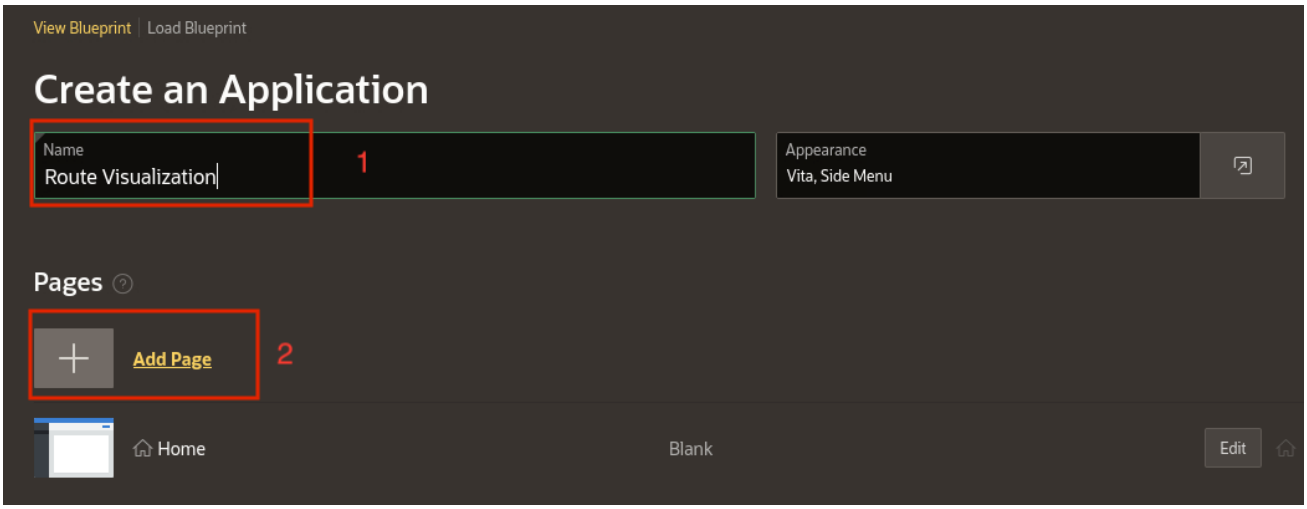


Escoja crear una nueva aplicación (New Application).

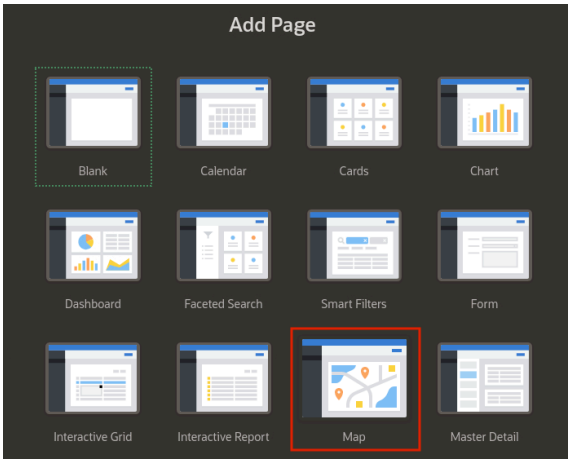
Copyright © 2022, Oracle and/or its affiliates. All rights reserved.



Introduzca un nombre para la aplicación, por ejemplo Route Visualization (1) y haga click en el botón Add Page para añadir una página a la aplicación.

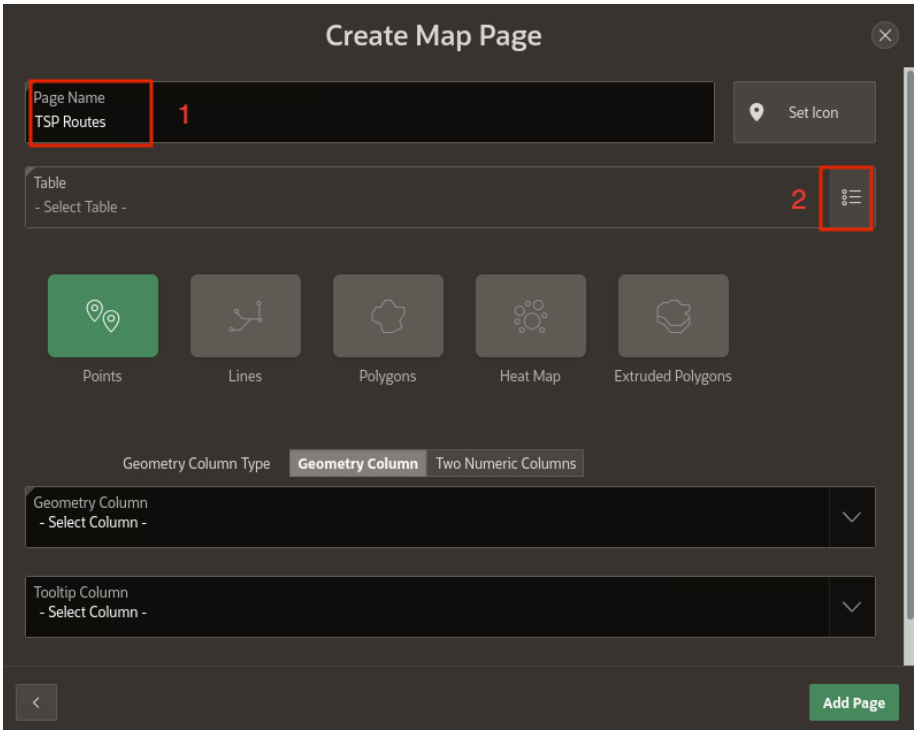


Escoja la plantilla de mapas en la parte inferior.

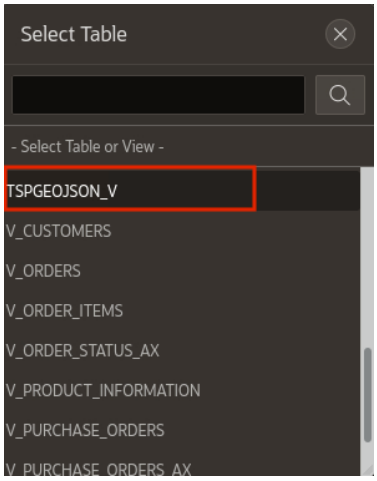


Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

El asistente de creación de página de mapa se muestra. Introduzc un título para la página, por ejemplo TSP Routes (1) y pulse el botón se selección de tabla (2) desde la que leer los datos a presentar en el mapa.

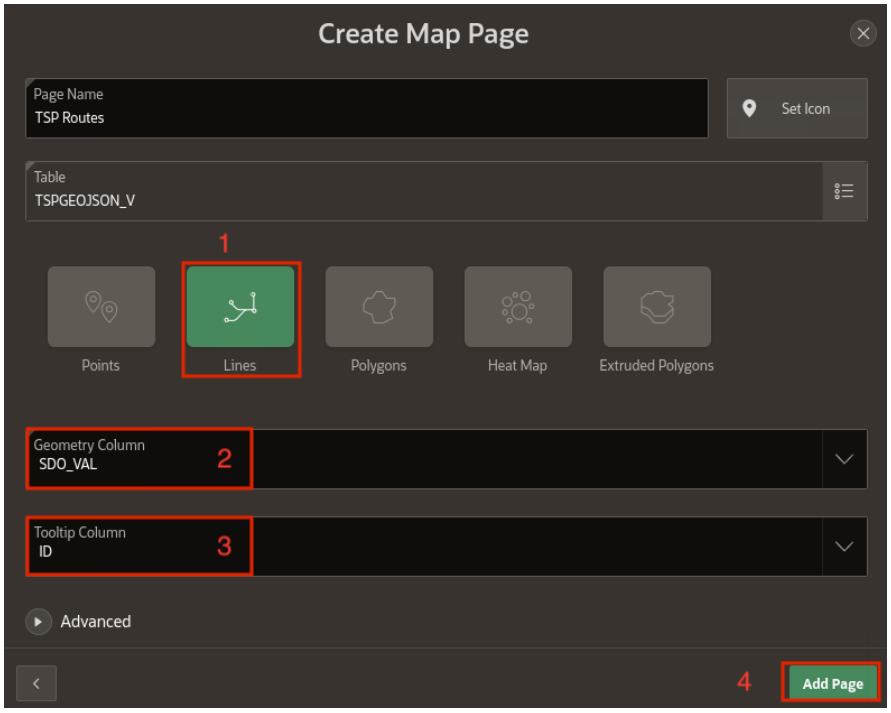


En el menú de selección de tabla, escoja la vista TSPGEOJSON_V creada anteriormente.

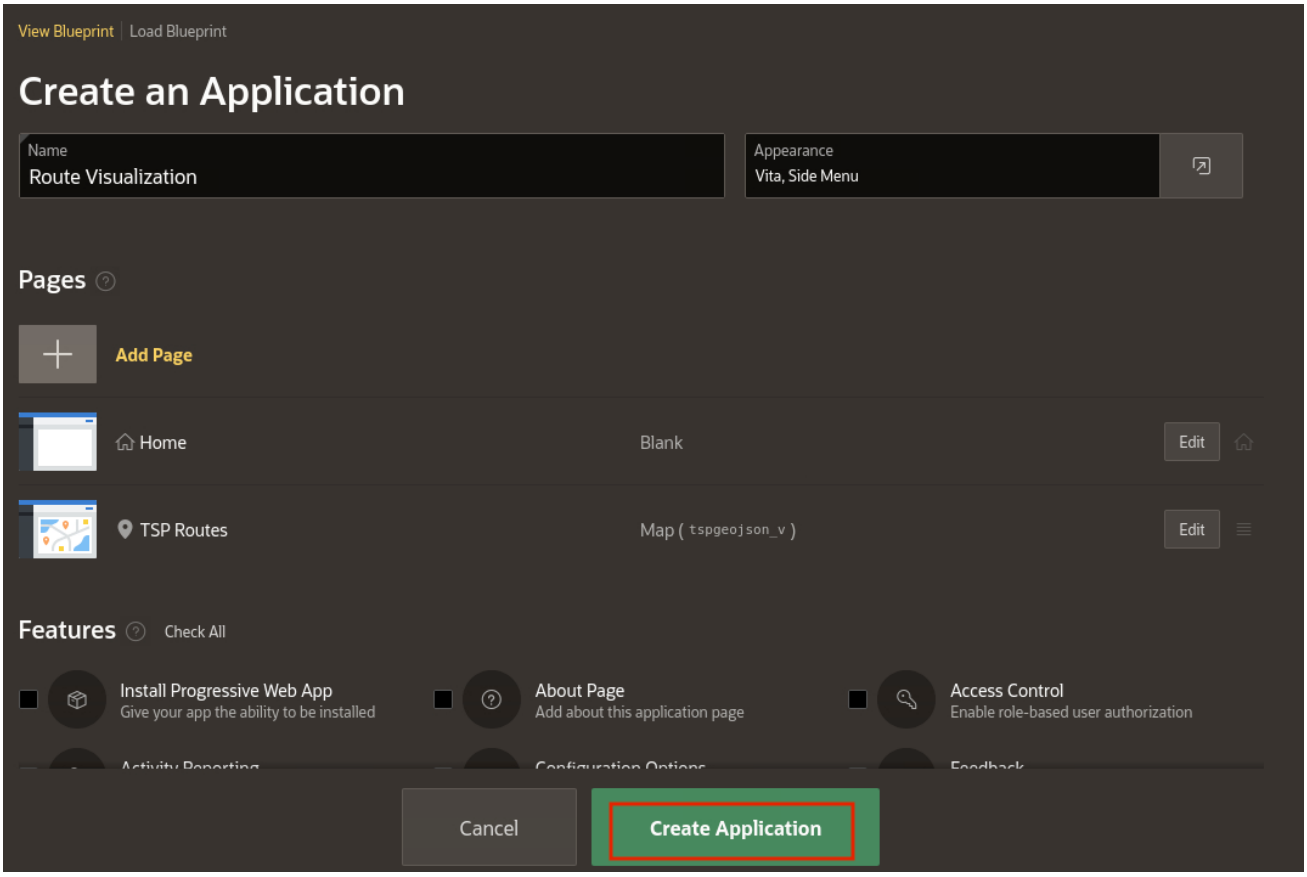


Especifique que lo que va a mostrar el mapa son líneas (1) que representan las rutas.
Escoja la columna de la vista que contiene las geometrías de las líneas: SDO_VAL (2).
Escoja una columna para usar como *tooltip* cuando se muestre la ruta, por ejemplo ID (3).
Para terminar el asistente, pulse el botón Add Page (4) que cerrará el asistente y le llevará de nuevo a la pantalla principal de la aplicación, donde podrá ver la nueva página en la lista de páginas.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

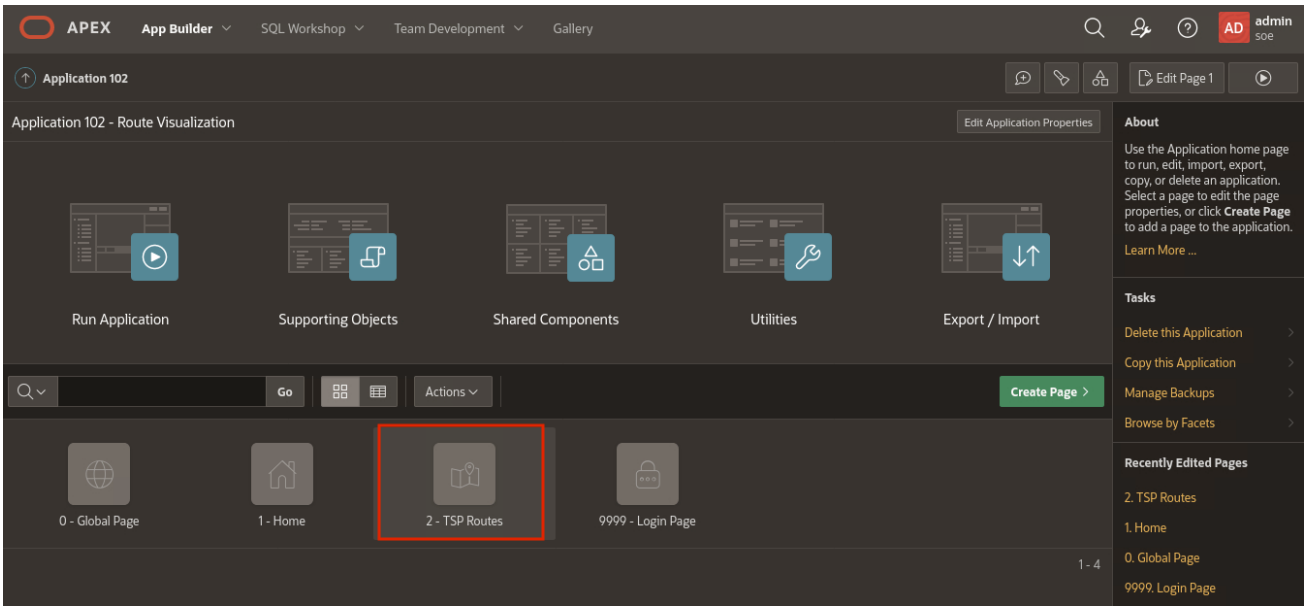


Este es el aspecto que tiene la aplicación con la nueva página de mapas. Haga click en el botón Create Application para crear la nueva aplicación Route Visualization.

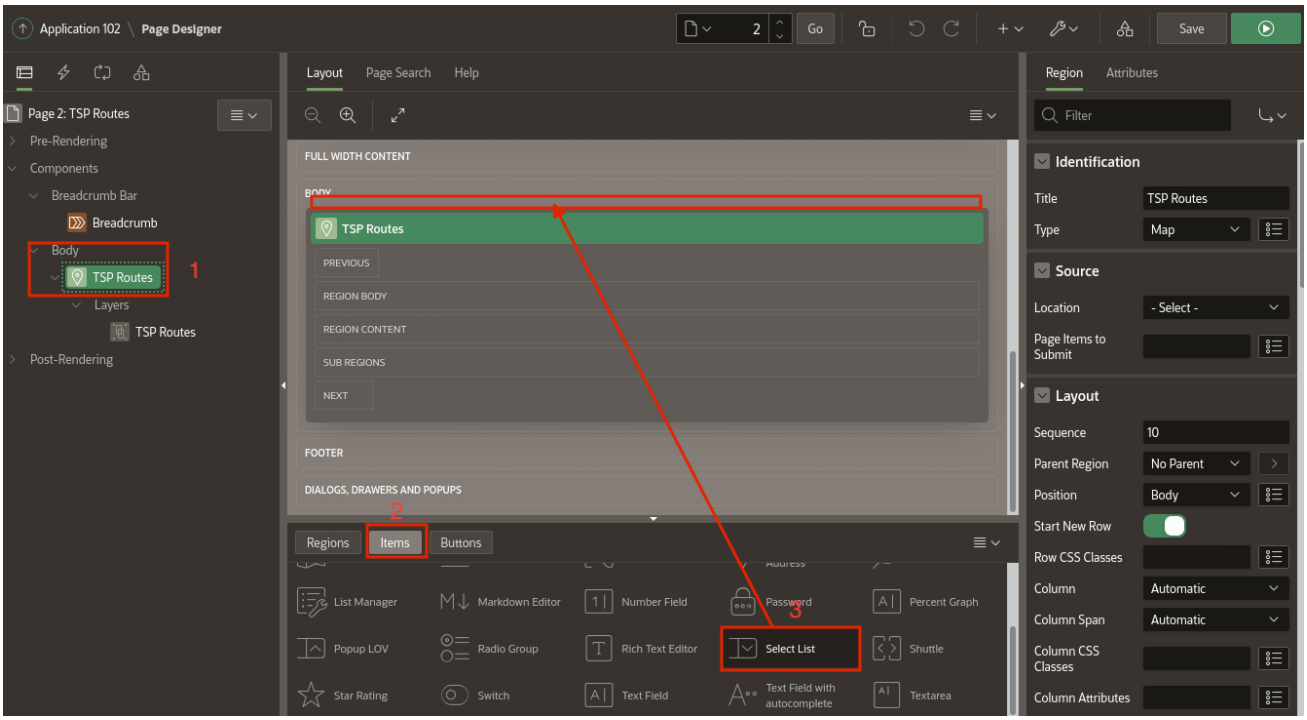


En la pantalla de la aplicación, seleccione la página 2-TSP Routes para realizar algunas mejoras sobre la plantilla.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

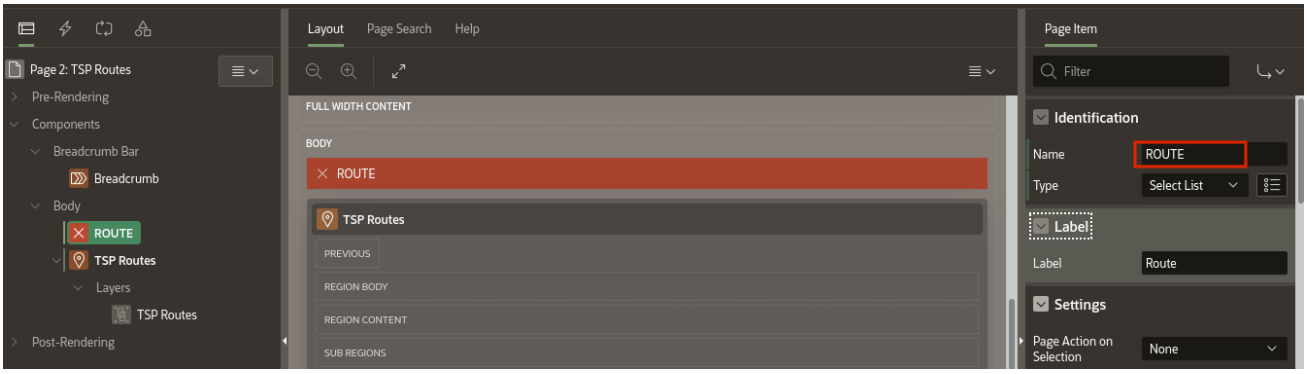


Se muestra el Page Designer, con todas las opciones disponibles. En el panel de la izquierda seleccione el componente del mapa TSP Routes (1). Esto mostrará en el panel central su posición en la página, dentro del BODY. Desde la paleta de Items (2) de la parte inferior, arrastre el componente Select List (3) y colóquelo en el BODY justo encima de TSP Routes.
Esto mostrará una lista desde la que seleccionar la ruta a pintar en el mapa.



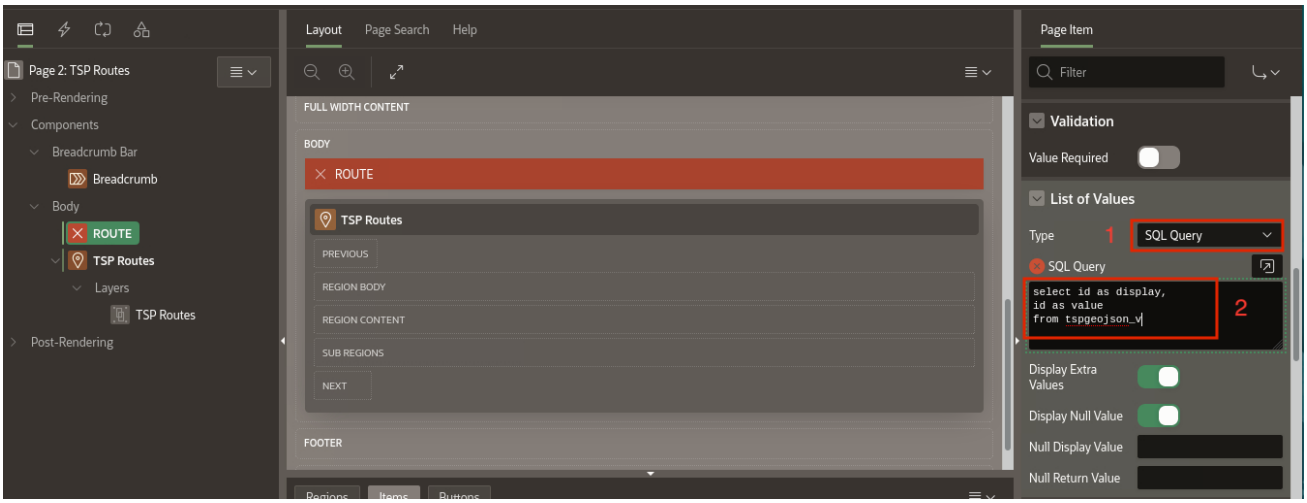
El nuevo componente se mostrará en el árbol del panel izquierdo así como en el BODY justo encima del componente del mapa TSP Routes.
En la derecha se encuentran las opciones de configuración de los componentes. En primer lugar se le da un nombre, por ejemplo ROUTE. Este nombre se usará más adelante para referirse al valor que haya seleccionado el usuario en este componente.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

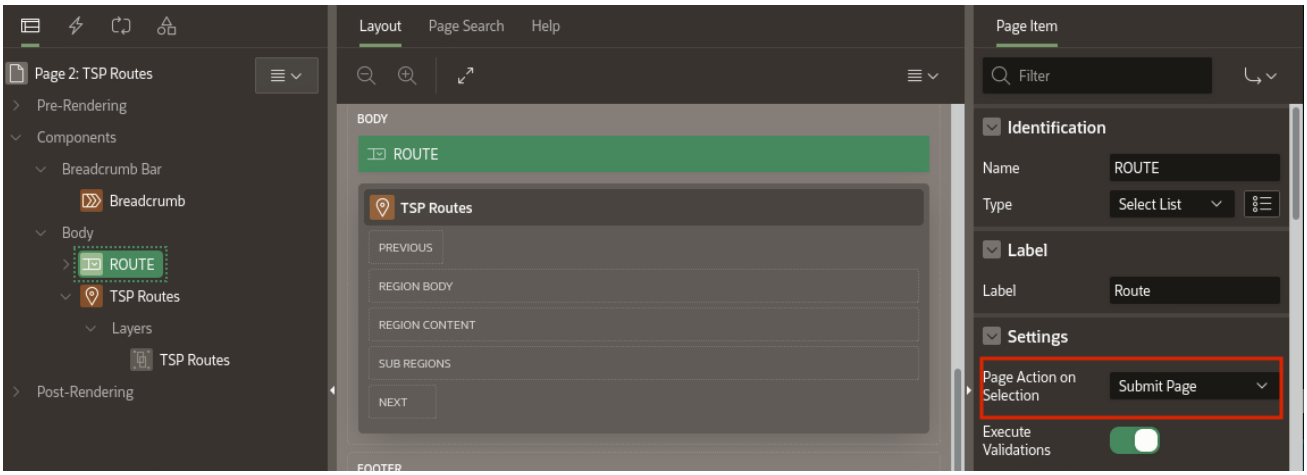


Más abajo en la configuración, busque List of Values, donde es necesario especificar los nombre y valores de la lista de selección. Escoja SQL Query (1) en Type, ya que se van a calcular dinámicamente desde la información de la base de datos. En la caja de texto SQL Query (2) introduzca la siguiente consulta:

```
select id as display, id as value from tspgeojson_v
```

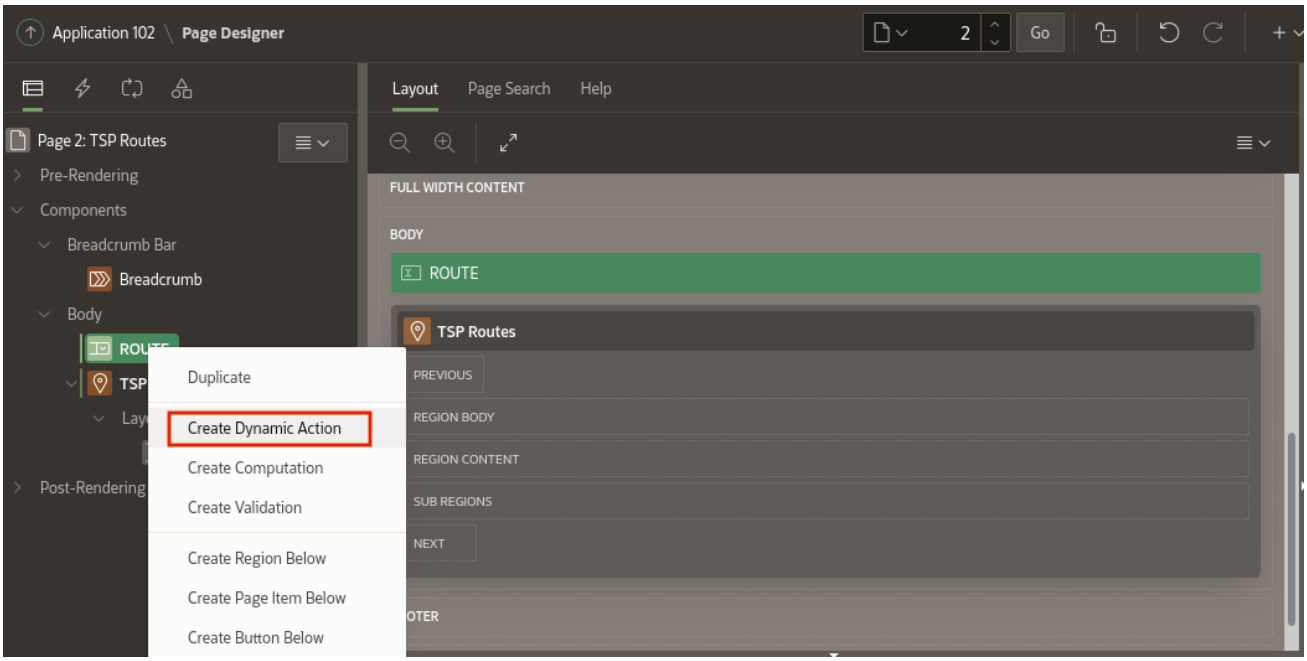


En la parte superior de las configuración, en **Settings**, escoja **Submit Page** como **Page Action on Selection** para que se envíe el valor de la selección en la variable ROUTE.



Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

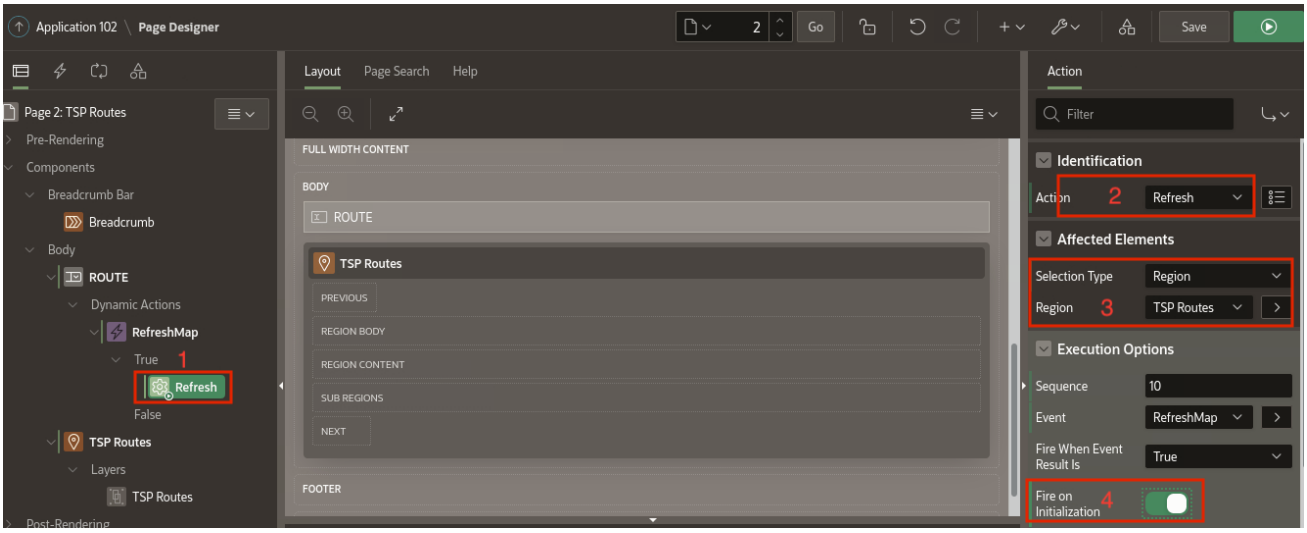
A continuación es necesario conectar la lista de selección con la visualización el mapa y forzar un refresco del mapa cada vez que se cambie la selección.
En el panel izquierdo, haga botón derecho sobre la lista de selección y escoja Create Dynamic Action.



De un nombre a la Dynamic Action, por ejemplo RefreshMap:

A continuación seleccione en el panel izquierdo la acción (1). En el panel derecho de configuración, especifique la acción que debe ocurrir cuando se haga una selección en la lista. El tipo de acción es **Refresh** (2) y debe aplicarse sobre una **Region** que es la del mapa **TSP Routes** (3). También especifique que la acción se realice cuando se inicialice la aplicación.

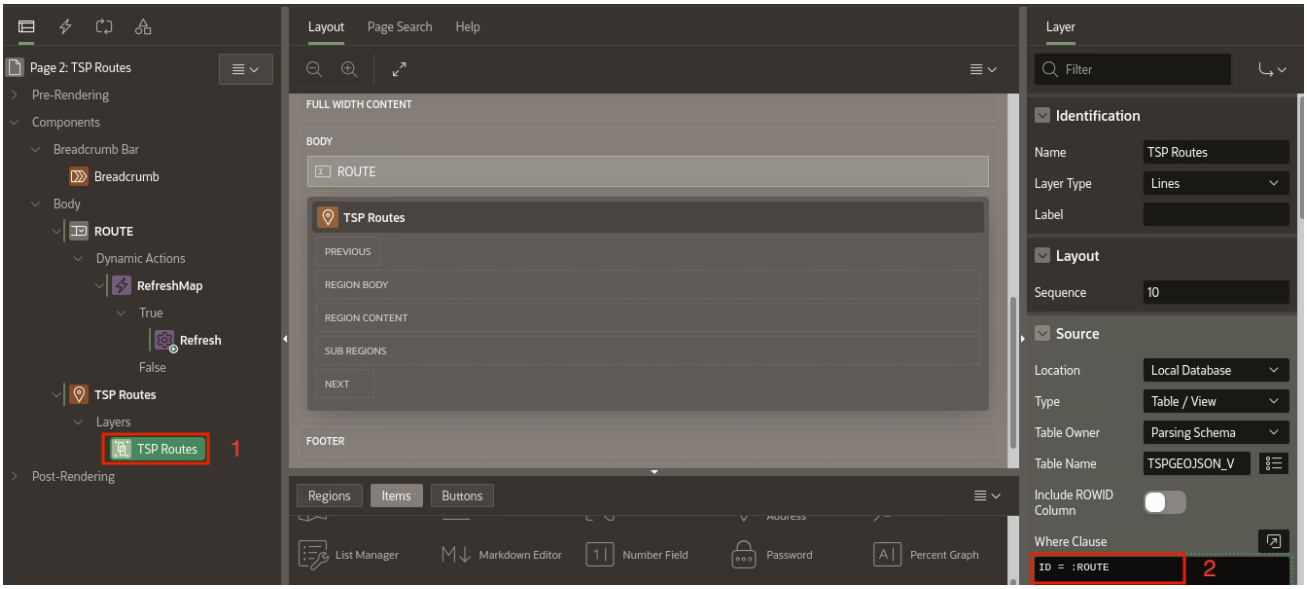
Copyright © 2022, Oracle and/or its affiliates. All rights reserved.



Escoja en el panel izquierdo la capa **TSP Routes**. En el panel derecho de configuración establecezca un filtro a partir del valor seleccionado en la lista de selección. Introduzca en la caja de texto **Where Clause** (2) lo siguiente:

```
ID = :ROUTE
```

De esta manera el valor de la lista de selección ROUTE establece la ruta a mostrar a partir de su ID. La forma de referenciarlo es añadiendo ":" delante de su nombre.

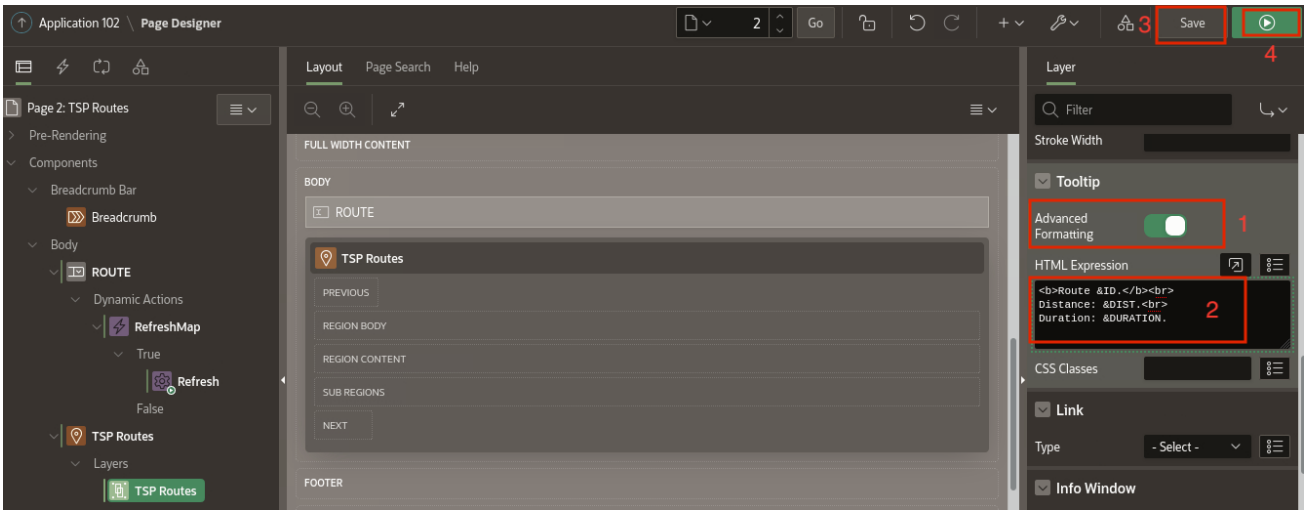


Por último, para mejorar la información que muestra el Tooltip, seleccione **Advanced Formatting** (1) y en la caja de texto **HTML Expression** (2) introduzca lo siguiente:

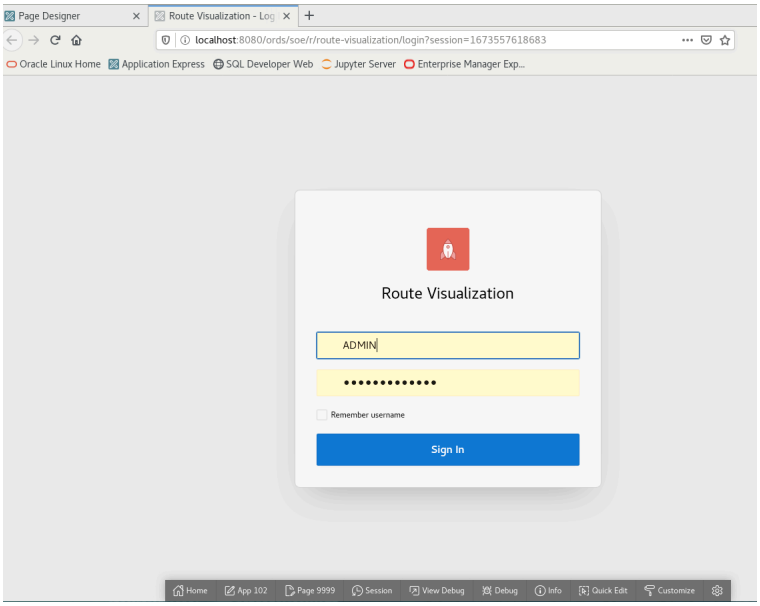
```
<b>Route &ID.</b><br>
Distance: &DIST.<br>
Duration: &DURATION.
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

La referencia a la información que viene de las columnas de la tabla se hace prefijando con “&” y con el sufijo “.”.

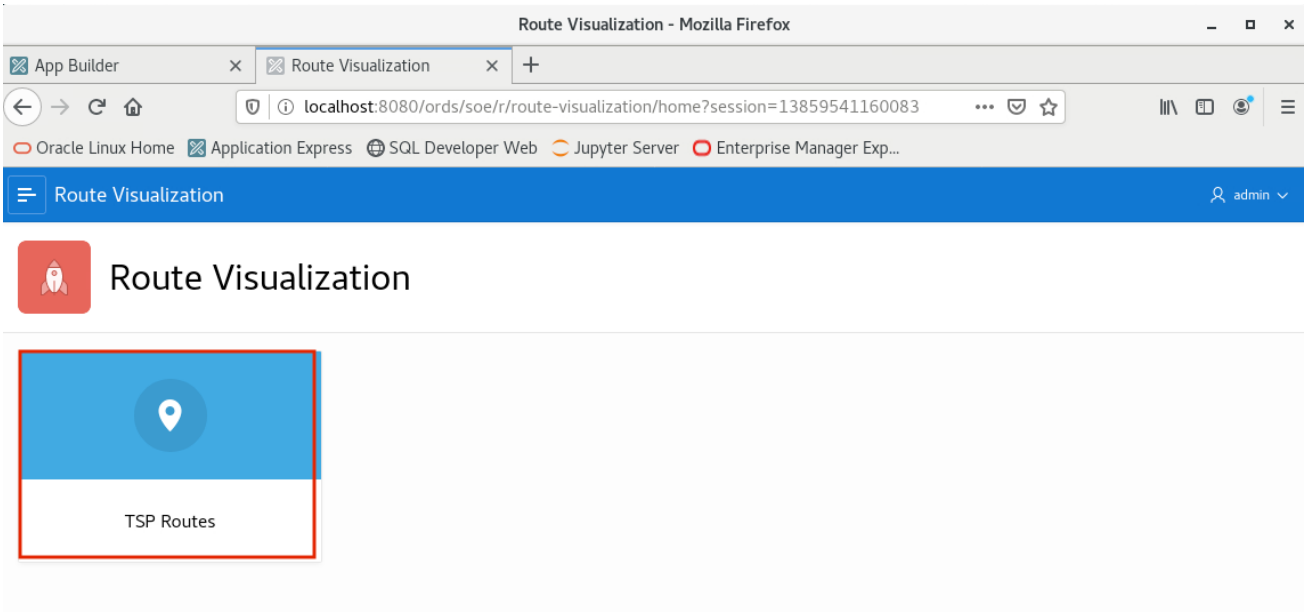


Pulse el botón **Save** (3) para guardar todos los cambios realizados.
Pulse el botón **Run** (4) para probar la aplicación.

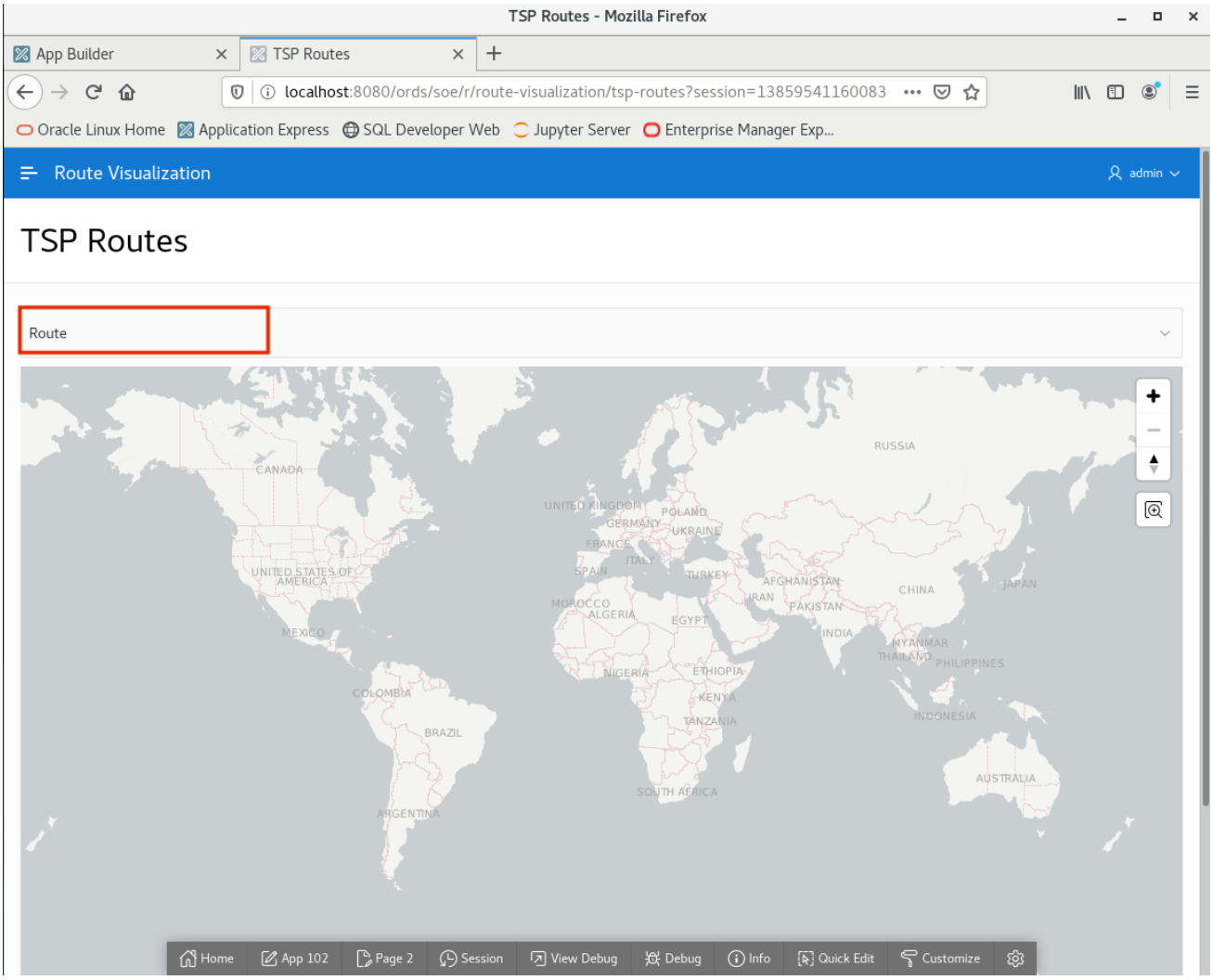


La aplicación aparecerá en una nueva pestaña del navegador donde hay que autenticarse para abrir una sesión. Use los datos se usuario y clave que le propone el navegador.
Una vez dentro de la aplicación haga click en el visualizador de mapa TSP Routes o bien en el menú de hamburguesa seleccione esta misma opción.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

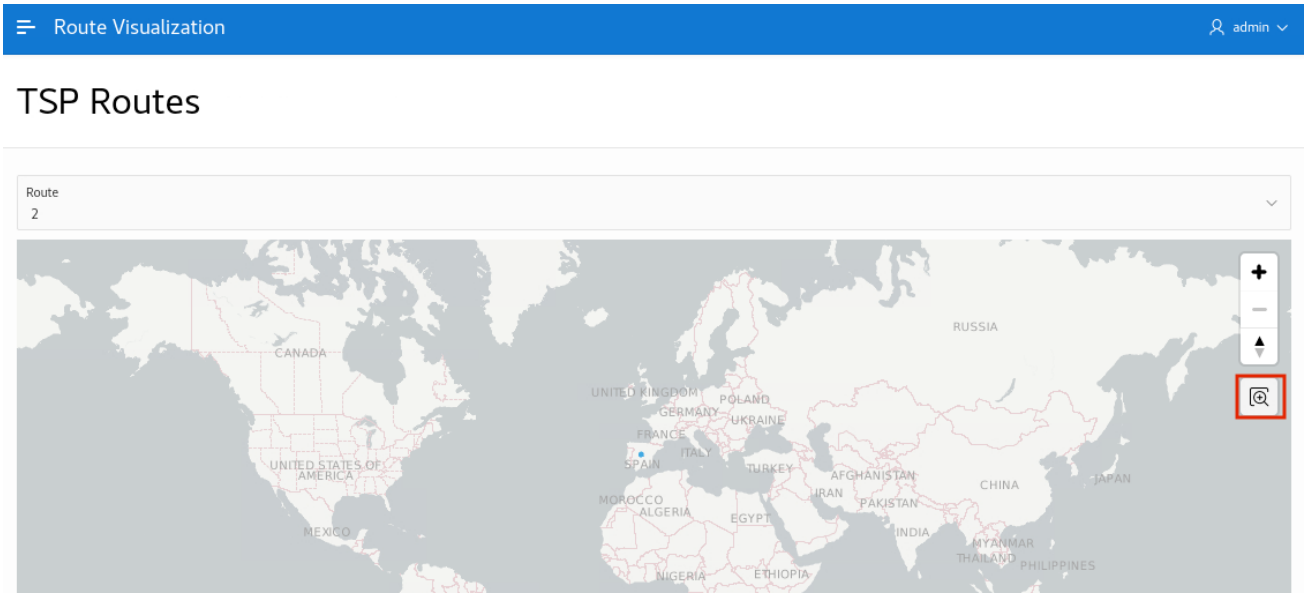


Se le presentará un mapa vacío. En la lista de selección, despliegue y escoja una de las rutas disponibles para que se muestre en el mapa.

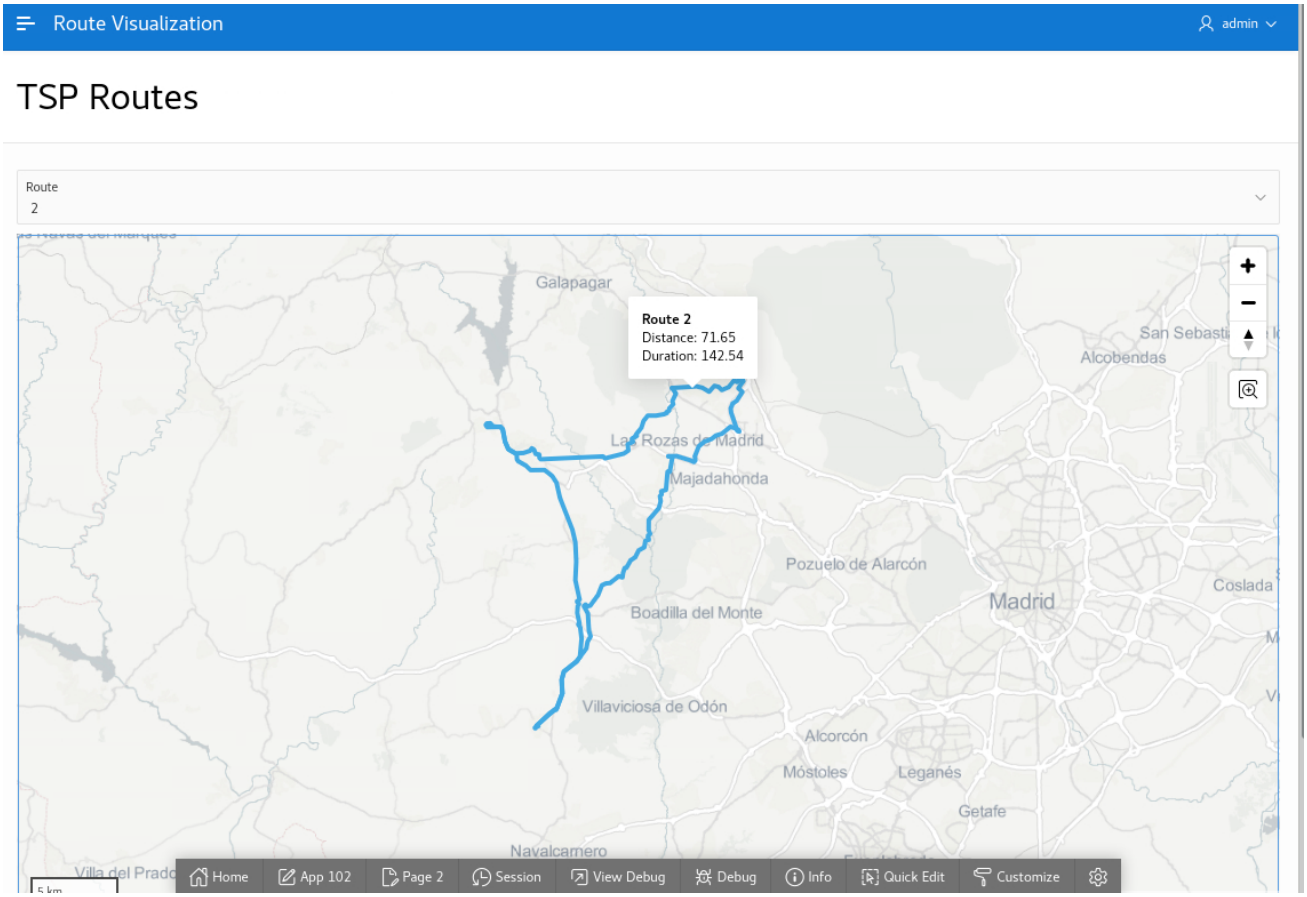


Use la herramienta de zoom sobre la Península Ibérica, si es necesario, para ver la ruta con detalle.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.



Al posicionar el ratón sobre la ruta se mostrará la información del Tooltip.



Enhorabuena, ha completado esta sección del taller.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

Creación de un recomendador usando *property graph* o grafo.

En esta actividad va a crear un grafo a partir de los datos de pedidos que existen en la PDB2 y los datos de clientes y productos que existen en la PDB1. Con este grafo se implementará un recomendador de productos.

Un grafo modela los datos en entidades (nodos) y relaciones/interacciones (aristas) entre las entidades. Cada nodo tiene un tipo y puede tener una cantidad opcional de propiedades en forma clave-valor. Cada arista tiene un tipo y puede tener una cantidad opcional de propiedades en forma clave-valor.

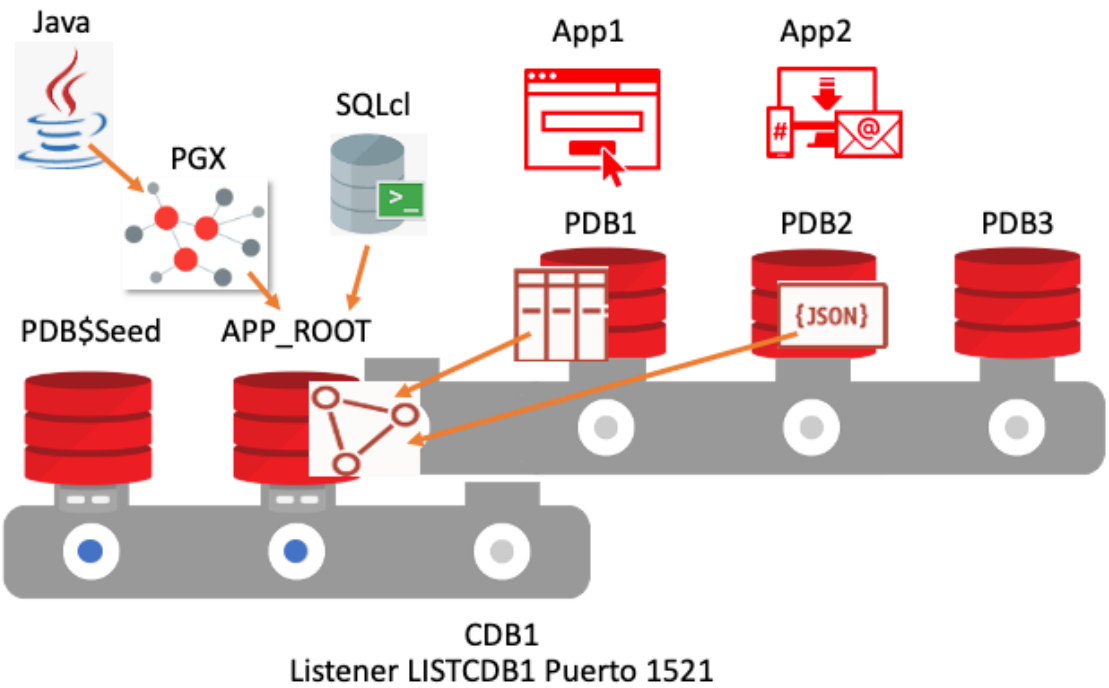
Para este ejercicio, se va a generar un grafo siguiendo el siguiente esquema:

Esta información reside en dos PDB diferentes:

1. Los clientes están en la PDB1 en la tabla CUSTOMERS
2. Los pedidos están en la colección SODA de PDB2
3. Los detalles de los productos están en la PDB1 en la tabla PRODUCT_INFORMATION

Desde el application container APP_ROOT se crearán unas vistas que permitan acceder a esta información de forma adecuada para construir el grafo. La información se mantiene en las PDB originales, todo este proceso tiene lugar de forma transparente a las aplicaciones que usan esas tablas o colecciones.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.



Estas vistas se usarán para definir proveedores de instancias o nodos y proveedores de relaciones o aristas en el grafo.

La creación del grafo se va a realizar con el lenguaje PGQL, que describirá qué tablas o vistas se transforman en nodos o aristas. Toda la información acerca de este lenguaje en https://pgql-lang.org/.

Para poder ejecutar PGQL en la base de datos es necesario tener un intérprete que lo soporte. En este caso se va a usar en primer lugar la nueva línea de comando SQLcl con el plugin de PGQL. Esta herramienta se encuentra instalada en el entorno de trabajo así como el plugin de PGQL. En segundo lugar se usará un programa Java para ejecutar una consulta PGQL que implementa un recomendador de productos básico. Este programa Java interactúa con el motor analítico PGX que carga la información del grafo desde la base de datos a memoria.

Creación de vistas en APP_ROOT

El usuario `soe` de APP_ROOT necesita disponer de los privilegios necesarios para poder construir grafos.

```
❯ sqlplus sys/Oracle_4U@localhost:1521/app_root as sysdba

SQL*Plus: Release 21.0.0.0.0 - Production on Tue Jan 11 12:40:43 2022
Version 21.3.0.0.0

Copyright (c) 1982, 2022, Oracle.  All rights reserved.


Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> grant graph_developer to soe;
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
Grant succeeded.

SQL> exit
```

En el usuario `soe` de APP_ROOT se crean las vistas necesarias sobre la información que reside en las pdbs: PDB1 y PDB2.

En el caso de PDB1 ya existe una vista que se creó en el día 1 de este workshop que es perfectamente válida: V_CUSTOMERS, que accede a la tabla CUSTOMERS.
Se va a crear una nueva vista sobre PRODUCT_INFORMATION. Para que la vista pueda acceder al contenido de la tabla PRODUCT_INFORMATION de la PDB1, debe existir una tabla vacía igual en APP_ROOT.

```
❯ /home/oracle/sqlcl/bin/sql soe/soe@localhost/app_root


SQLcl: Release 21.2 Production on Tue Feb 22 09:17:20 2022

Copyright (c) 1982, 2022, Oracle.  All rights reserved.

Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0


SQL> CREATE TABLE PRODUCT_INFORMATION
       (   PRODUCT_ID NUMBER(6,0),
      PRODUCT_NAME VARCHAR2(50 BYTE) ,
      PRODUCT_DESCRIPTION VARCHAR2(2000 BYTE),
       CATEGORY_ID NUMBER(4,0),
       WEIGHT_CLASS NUMBER(1,0),
       WARRANTY_PERIOD INTERVAL YEAR (2) TO MONTH,
       SUPPLIER_ID NUMBER(6,0),
       PRODUCT_STATUS VARCHAR2(20 BYTE),
      LIST_PRICE NUMBER(8,2),
      MIN_PRICE NUMBER(8,2),
     CATALOG_URL VARCHAR2(50 BYTE));

Table PRODUCT_INFORMATION created.

SQL> create or replace view V_PRODUCT_INFORMATION as select * from
containers(PRODUCT_INFORMATION);

View V_PRODUCT_INFORMATION created.
```

En el caso de PDB2, la información de los pedidos se encuentra en los documentos JSON de la colección SODA PURCHASE_ORDERS. Se construirán dos vistas sobre estos documentos JSON para extraer por un lado los pedidos y por otro lado los elementos de cada pedido.

En este caso ya existe en el APP_ROOT una colección SODA llamada PURCHASE_ORDERS en APP_ROOT que se creó en el día 1 del taller, entonces se puede directamente crear las vistas.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
SQL> create or replace view V_ORDERS as
    select j.JSON_DOCUMENT.ORDER_ID.number() as ORDER_ID,
           j.JSON_DOCUMENT.CUSTOMER_ID.number() as CUSTOMER_ID,
           j.JSON_DOCUMENT.ORDER_DATE.string() as ORDER_DATE,
           j.JSON_DOCUMENT.ORDER_TOTAL.string() as ORDER_TOTAL,
           j.JSON_DOCUMENT.ORDER_MODE.string() as ORDER_MODE
   from containers(PURCHASE_ORDERS) j;

View V_ORDERS created.

SQL> create or replace view V_ORDER_ITEMS as
    select ORDER_ID || LINE_ITEM_ID id, ORDER_ID, PRODUCT_ID,QUANTITY, UNIT_PRICE
    from
    (select jt.*
    from containers(PURCHASE_ORDERS),
         JSON_TABLE (JSON_DOCUMENT, '$' COLUMNS (
         ORDER_ID number       path '$.ORDER_ID.number()',
         NESTED PATH '$.ITEMS[*]'  COLUMNS(
         PRODUCT_ID  number      path '$.PRODUCT_ID.number()',
         LINE_ITEM_ID number     path '$.LINE_ITEM_ID.number()',
         QUANTITY   number       path '$.QUANTITY.number()',
         UNIT_PRICE number       path '$.UNITPRICE.number()'
         )))jt);

View V_ORDER_ITEMS created.
```

Recapitulando, las vistas creadas en APP_ROOT en este punto son las siguientes.

```
SQL> select view_name from user_views;

               VIEW_NAME
_________________________
V_PRODUCT_INFORMATION
V_ORDERS
V_ORDER_ITEMS
V_CUSTOMERS
V_PURCHASE_ORDERS
V_ORDER_STATUS_AX
V_PURCHASE_ORDERS_AX
```

Las 4 primeras son las que se emplearán para crear el grafo en la siguiente sección.

Creación de grafo en APP_ROOT

La creación del grafo se hará con lenguaje PGQL, hay que activar este lenguaje en SQLcl con la siguiente sentencia:

```
SQL> pgql auto on

PGQL Auto enabled for graph=[null], execute=[true], translate=[false]
SQL>
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

A partir de este momento se puede usar lenguaje PGQL. Con la siguiente sentencia PGQL se va a construir el grafo:

```
SQL> create property graph GRAFO1
   vertex tables (
       V_CUSTOMERS key (CUSTOMER_ID)
         Label CUSTOMER
         Properties (CUSTOMER_ID,CUST_FIRST_NAME, CUST_LAST_NAME),
       V_ORDERS key (ORDER_ID)
         Label ORDER
         Properties (ORDER_ID,ORDER_DATE,ORDER_MODE,ORDER_TOTAL),
       V_PRODUCT_INFORMATION key (PRODUCT_ID)
         Label PRODUCT
         Properties (PRODUCT_ID,PRODUCT_NAME,PRODUCT_DESCRIPTION, LIST_PRICE)
   )
   edge tables (
      V_ORDERS as CUST_ORD key (ORDER_ID)
        Source key (CUSTOMER_ID) references V_CUSTOMERS
        Destination key (ORDER_ID) references V_ORDERS
        Label HAS_ORDER
        NO PROPERTIES,
      V_ORDER_ITEMS key (id)
        Source key (ORDER_ID) references V_ORDERS
        Destination key (PRODUCT_ID) references V_PRODUCT_INFORMATION
        Label HAS_PRODUCT
        PROPERTIES (ORDER_ID,PRODUCT_ID,UNIT_PRICE,QUANTITY)
  ) OPTIONS (PG_VIEW);

Graph created
```

Esta sentencia usa las vistas V_CUSTOMERS, V_ORDERS y V_PRODUCT_INFORMATION para crear un vértice o nodo por cada una de sus filas. Algunas de las columnas de estas vistas se convierten en propiedades del nodo. Por ejemplo cada fila de la vista V_CUSTOMERS se va a convertir en un nodo de tipo CUSTOMER que contiene tres propiedades: CUSTOMER_ID, CUST_FIRST_NAME y CUST_LAST_NAME.

Por otro lado las vistas V_ORDERS y V_ORDER_ITEMS se usan para crear las aristas entre los nodos. También pueden tener propiedades al igual que los nodos. En este caso se ha de especificar qué columna se debe usar para buscar el tipo de nodo origen de la arista y cual para buscar el tipo de nodo destino. Por ejemplo, V_ORDERS creará aristas del tipo HAS_ORDER; usará la columna CUSTOMER_ID para identificar un nodo de tipo CUSTOMER que será el origen y la columna ORDER_ID para identificar un nodo de tipo ORDER que será el destino.

Al ejecutar esta sentencia, los metadatos del grafo se almacenan en un conjunto de nuevas tablas que se crean en el propio esquema `soe` y cuyo nombre tiene como prefijo el nombre del grafo.

```
SQL> pgql auto off

PGQL Auto disabled

SQL> select table_name from user_tables
     where table_name like 'GRAFO1%';

             TABLE_NAME
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
_______________________
GRAFO1_ELEM_TABLE$
GRAFO1_KEY$
GRAFO1_LABEL$
GRAFO1_PROPERTY$
GRAFO1_SRC_DST_KEY$
```

Estas tablas únicamente contienen metadados, los datos residen en las vistas a las que hizo referencia la sentencia '`CREATE PROPERTY GRAPH`'.

Pruebe algunas consultas PGQL con el grafo recién creado desde SQLcl.

```
SQL> pgql auto on graph GRAFO1

PGQL Auto enabled for graph=[GRAFO1], execute=[true], translate=[false]

SQL> select op.QUANTITY, p.LIST_PRICE, p.PRODUCT_NAME
    from match (c)-[co]->(o:ORDER)-[op]->(p:PRODUCT)
    where c.CUST_FIRST_NAME='john' and c.CUST_LAST_NAME='johnson'
    limit 5;

   QUANTITY    LIST_PRICE                          PRODUCT_NAME
___________ _____________ ____________________________________
          5          3722 9jRvUKRhPrqQkSl
          5          3331 AKv9Tb9jMDfRMllD NCODtNP9fPJT mVWV
          4          2678 eEWjYC l2O7nU6
          4          4125 T1p0H3PAklCRXE6b
          5          1632  Fi8GNZ2XJUSslzdD rlT6
```

El lenguaje PGQL es similar a SQL, pero usa la cláusula MATH para identificar nodos: (n) y aristas –[e]-> que pueden referenciarse por alias. También se pueden incluir las propiedades de los nodos y aristas en las consultas.

Acceso al grafo desde Java

A continuación se va a realizar una consulta al grafo desde lenguaje Java. Esta consulta la resuelve la base de datos, pero es necesario crear el intérprete de PGQL. El resultado de la consulta es recogido en forma de un ResultSet muy parecido al de las consultas SQL.

Suponga que tiene identificado un cliente modelo con CUSTOMER_ID=1 y desea comparar sus pedidos con los del resto de clientes, buscando aquellos otros clientes que tienen la mayor cantidad de productos en común con este cliente modelo. De esta manera se puede implementar un recomendador muy simple.

Para ello puede usar dos cláusulas MATCH en las que la primera de ellas: MATCH (c:CUSTOMER)->(:ORDER)-[:HAS_PRODUCT]->(p:PRODUCT) busca los productos (p) del cliente (c) donde c es el cliente modelo y a partir de esos productos, se realiza un segundo match en el sentido contrario, buscando aquellos clientes (c1) que tienen la mayor cantidad de productos en común con el cliente c.

La consulta que se va a usar es la siguiente:

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
select c1.CUST_FIRST_NAME, c1.CUST_LAST_NAME
FROM MATCH (c:CUSTOMER)->(:ORDER)-[:HAS_PRODUCT]->(p:PRODUCT),
     MATCH (c1:CUSTOMER)->(:ORDER)-[:HAS_PRODUCT]->(p:PRODUCT)
WHERE c.CUSTOMER_ID=1 AND c.CUSTOMER_ID <> c1.CUSTOMER_ID
GROUP BY c1 ORDER BY count(DISTINCT p) DESC LIMIT 10;
```

Esta consulta se va a realizar desde un programa Java, que imprime el resultado por pantalla. El código de la clase está en el repositorio de Github usado en la sesión anterior.
En caso de no tenerlo disponible, vuelva a clonarlo:

```
❯ git clone https://github.com/OracleDataManagementSpain/ConvergedDatabase

Cloning into 'ConvergedDatabase'...
remote: Enumerating objects: 658, done.
remote: Counting objects: 100% (149/149), done.
remote: Compressing objects: 100% (113/113), done.
remote: Total 658 (delta 69), reused 64 (delta 33), pack-reused 509
Receiving objects: 100% (658/658), 403.12 MiB | 23.26 MiB/s, done.
Resolving deltas: 100% (283/283), done.
```

Entre a la carpeta ConvergedDatabase/Developers, donde existe un fichero PgqlQuery.java:

```
❯ cd ConvergedDatabase/Developers
❯ ls
AEMETRequest.java  java_libraries.zip  nodeapp2.zip  PgqlQueryPGX.java  Readme.md
```

Por simplicidad de esta actividad las credenciales de conexión se encuentran en las primeras líneas de código. Esto es una práctica no recomendada en un sistema real, pero por simplicidad se ha usado.

```
  public static void main(String[] args) throws Exception
  {
    int idx=0;
    String host               = "localhost";
    String port               = "7007";
    String sid                = "app_root";
    String user               = "soe";
    String password           = "soe";
    String graph              = "GRAFO1";
```

Con estas credenciales se conecta al motor de grafos y se lee el grafo creado:

```
System.out.println("Conectando...");
ServerInstance instance =
GraphServer.getInstance("https://"+host+":"+port,user,password.toCharArray());
session = instance.createSession("developers-workshop-session");

System.out.println("Leyendo datos...");
g = session.readGraphByName(graph,GraphSource.PG_VIEW);
```

Con esa conexión se ejecuta la sentencia, se recupera el resultado y se imprime por salida estándar.

```
String pgql = "select c1.CUST_FIRST_NAME, c1.CUST_LAST_NAME "+
              "FROM MATCH (c:CUSTOMER)->(:ORDER)-[:HAS_PRODUCT]->(p:PRODUCT), "+
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
                    "MATCH (c1:CUSTOMER)->(:ORDER)-[:HAS_PRODUCT]->(p:PRODUCT) "+
               "WHERE c.CUSTOMER_ID=1 AND c.CUSTOMER_ID <> c1.CUSTOMER_ID  "+
               "GROUP BY c1 ORDER BY count(DISTINCT p) DESC LIMIT 10";

System.out.println("Ejecutando consulta...");
resultSet = g.queryPgql(pgql);

// Print the results
System.out.println("Lista de clientes a recomendar productos");
resultSet.print();
```

Compile la clase java usando el CLASSPATH donde están las librerías necesarias:

```
❯ javac -cp "/opt/oracle/graph/lib/*"  PgqlQueryPGX.java
```

Estas librerías están disponibles en la página de descargas de Oracle Graph Server en la siguiente dirección: https://www.oracle.com/database/technologies/spatialandgraph/property-graph-features/graph-server-and-client/graph-server-and-client-downloads.html

Una vez compilada la clase, ejecute para observar el resultado.

```
❯ java -cp "/opt/oracle/graph/lib/*:."  PgqlQueryPGX
Conectando...
Leyendo datos...
Ejecutando consulta...
Lista de clientes a recomendar productos
+----------------------------------+
| CUST_FIRST_NAME | CUST_LAST_NAME |
+----------------------------------+
| clifford        | mason          |
| franklin        | willis         |
| orlando         | benson         |
| ralph           | morgan         |
| rick            | hansen         |
| chad            | ellis          |
| jesus           | ruiz           |
| leo             | rose           |
| orlando         | benson         |
| john            | johnson        |
+----------------------------------+
```

Este listado muestra los 10 clientes que tienen pedidos muy parecidos al cliente modelo, a los que se les puede recomendar los productos que no tienen en común con él.

Enhorabuena, ha terminado esta sección en la que ha creado a partir de sus datos de clientes, pedidos y productos un grafo que le permite generar recomendaciones de productos, que puede invocar desde Java.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

Modelo predictivo python in-database y API Rest

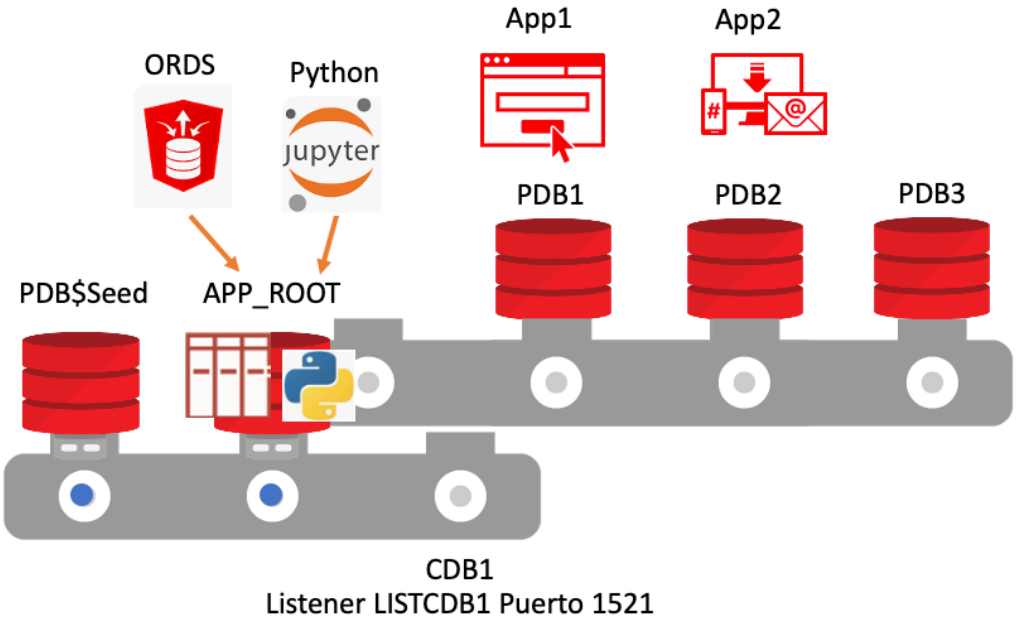
En esta sección se va a suponer que el equipo de científicos de datos de la empresa ha creado a partir de la información de pedidos, distancias y tiempos de reparto, datos meteorológicos y otros datos un juego de datos para entrenar un modelo de *machine learning* que predice si un pedido se entregará en la fecha prevista o por el contrario tendrá un retraso.

Para esta actividad se ha habilitado OML4Py en la PDB APP_ROOT, que permite la ejecución de lenguaje Python en esta PDB. Para trabajar de manera más cómoda con Python, se ha instalado también Jupyter, una herramienta muy popular entre los científicos de datos para la creación de *notebooks* desde los que desarrollar en este (y otros) lenguajes.

Una vez el modelo esté creado y almacenado en la base de datos, será posible ejecutarlo de diferentes maneras:

- Desde Python, usando Jupyter u otra herramienta
- Desde una conexión que permita ejecutar SQL
- Desde una llamada POST a ORDS (Oracle Rest Data Services)

El siguiente diagrama muestra los distintos componentes que van a participar en esta actividad.



Creación del modelo predictivo

El usuario `soe` de APP_ROOT necesita disponer de los privilegios necesarios para poder almacenar código Python en su esquema y hacer ejecución del mismo dentro de la base de datos.

```
> sqlplus sys/Oracle_4U@localhost:1521/app_root as sysdba

SQL*Plus: Release 21.0.0.0.0 - Production on Tue Jan 11 12:40:43 2022
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
Version 21.3.0.0.0

Copyright (c) 1982, 2022, Oracle.  All rights reserved.


Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> grant PYQADMIN to soe;

Grant succeeded.

SQL> exit
```

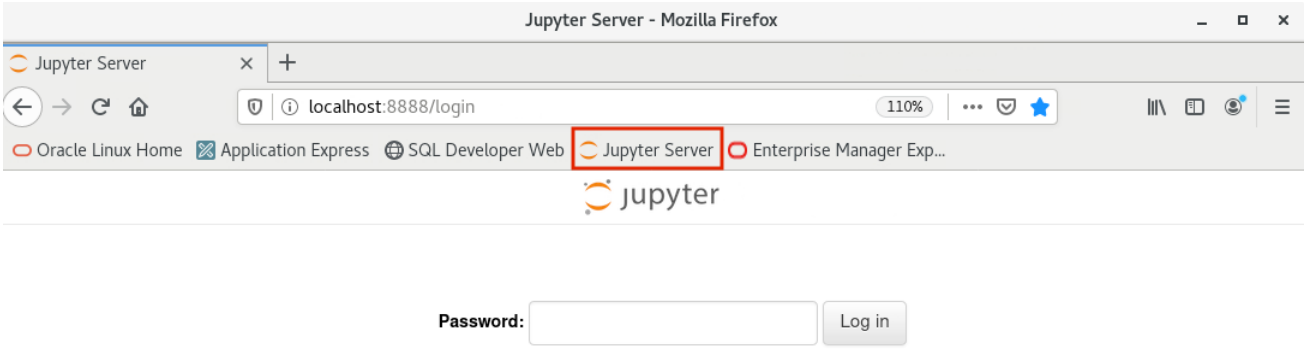
A continuación, se carga el tablón que ha sido desarrollado por los científicos de datos y que se encuentra en el fichero `/home/oracle/OML4Py/orders_ml_final.sql` del entorno del taller. En esta actividad, por simplicidad se facilita esta información en forma de un script de carga. Cargue el tablón en el esquema soe de APP_ROOT, creará la tabla necesaria para la ejecución del código Python en los siguientes pasos de la actividad.

```
 ❯ sqlplus soe/soe@localhost:1521/app_root

SQL*Plus: Release 21.0.0.0.0 - Production on Tue Jan 11 12:40:43 2022
Version 21.3.0.0.0

Copyright (c) 1982, 2022, Oracle.  All rights reserved.


Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> @/home/oracle/OML4Py/orders_ml_final.sql;

[…]

SQL> exit
```

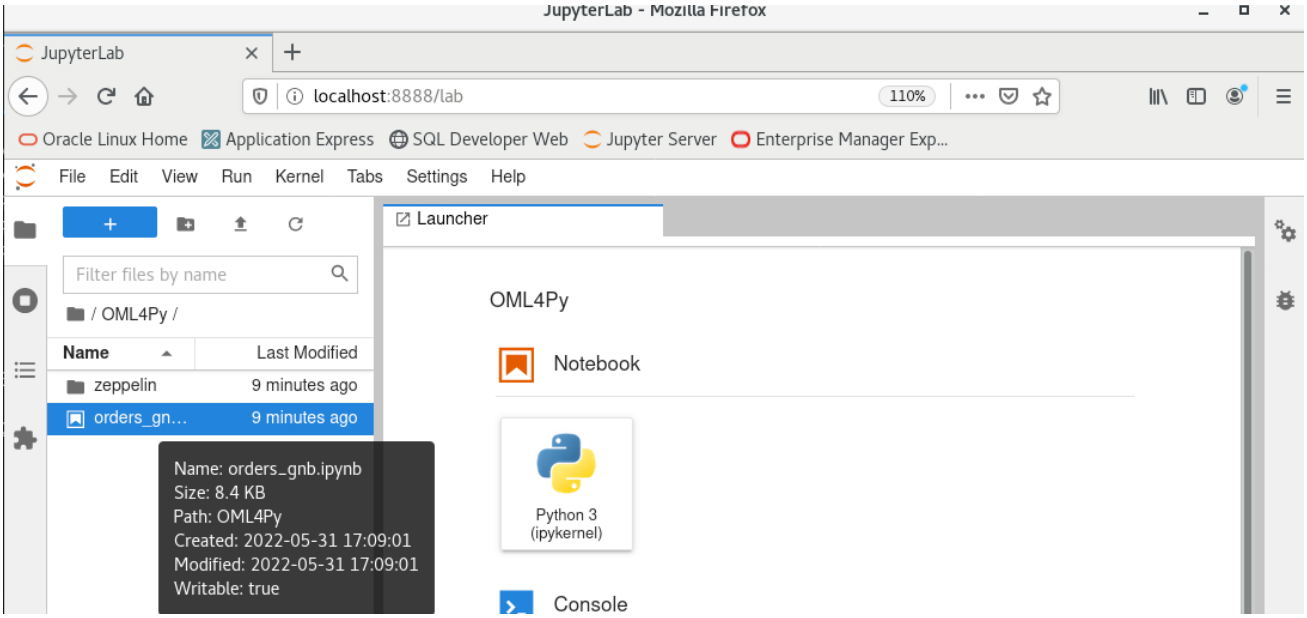
En el entorno del taller dispone de Jupyter. Esta herramienta está instalada en un usuario diferente del usuario `oracle`, ya que en un entorno real estaría instalada en un servidor diferente al de la base de datos o incluso localmente en el puesto de trabajo del científico de datos. Por simplicidad en este workshop se encuentra en la misma máquina que la base de datos.

Acceda a Jupyter usando el navegador Firefox y el marcador de la barra de marcadores.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

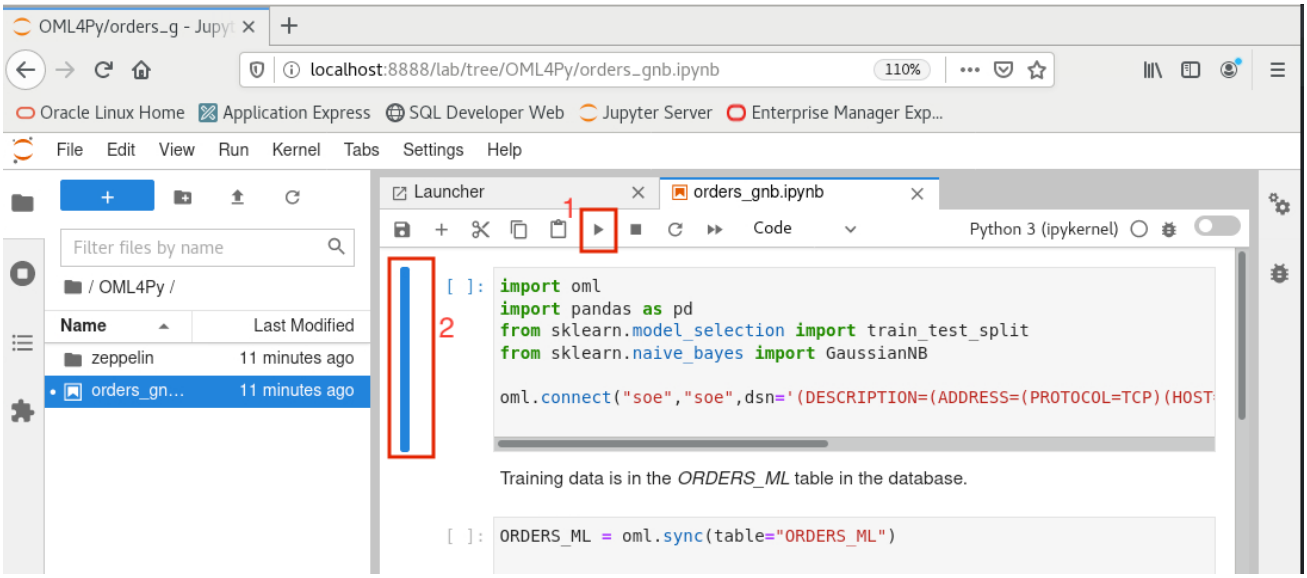


Acceda con la clave que le proporcionará el instructor.
En el panel de la izquierda hay un notebook que han preparado los científicos de datos llamado orders_gnb.pynb. Haga doble click sobre él para abrirlo.

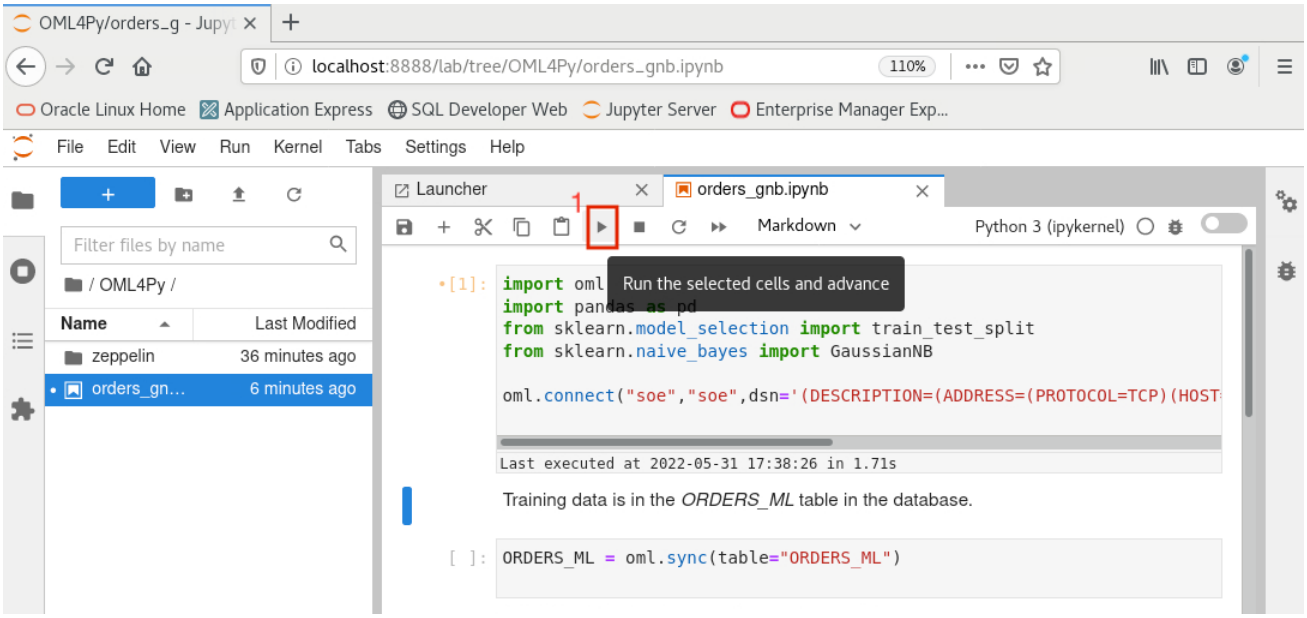


Una vez abierto el notebook, los párrafos y el contenido se mostrarán en el panel de la derecha. El notebook es un script secuencial que se ejecuta paso a paso con el botón de *play* de la barra superior de herramientas (1) y una barra vertical azul (2) indica qué párrafo va a ejecutarse.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.



A medida que se ejecutan los párrafos la barra azul va avanzando al siguiente, de manera que sucesivos click en el botón play (1) ejecutan en la secuencia correcta todo el código contenido en el notebook.



Los primeros párrafos cargan las librerías necesarias, entre ellas la de OML4Py, establecen una conexión a la base de datos usando el usuario SOE y la PDB APP_ROOT. En este esquema se encuentra el tablón preparado por los científicos de datos en la tabla ORDERS_ML. Para trabajar con este juego de datos se usa un objeto proxy que se construye con oml.sync.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
•[1]: import oml
      import pandas as pd
      from sklearn.model_selection import train_test_split
      from sklearn.naive_bayes import GaussianNB

      oml.connect("soe","soe",dsn='(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST
```

Last executed at 2022-05-31 17:38:26 in 1.71s

Training data is in the *ORDERS_ML* table in the database.

```
•[2]: ORDERS_ML = oml.sync(table="ORDERS_ML")
```

Last executed at 2022-05-31 17:47:13 in 13ms

Existe la posibilidad de cargar el tablón de entrenamiento desde la tabla ORDERS_ML en la base de datos a Jupyter y crear localmente el modelo predictivo. Esto puede hacerse por motivos de comodidad si ya existe un código desarrollado en Python y siempre que el volumen de datos en ORDERS_ML pueda ser cargado en la memoria de Jupyter. Esta posibilidad se muestra en la Opción 1 del notebook.

El modelo que se crea es un Naive Bayes Gausiano (GaussianNB), que una vez entrenado se almacena en el Data Store (oml.ds.save) de la propia base de datos, que se encuentra en el esquema SOE con el que se conectó al comienzo del notebook.

OPTION 1: Traditional method

Model can be created locally in this Jupyter session.

This means the data has to be pulled out of the database.

```
[ ]: orders = ORDERS_ML.pull()
     features = list(orders.columns[1:7])
     X=orders[features]
     y=orders['PRED']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
[ ]: gnb = GaussianNB()
     y_pred = gnb.fit(X_train, y_train).predict(X_test)
     print("Number of mislabeled points out of a total %d points : %d" % (X_test.shape[0], (y_test != y_
```

Now the model can be put in the database use schema's Data Store for later.

```
[ ]: oml.ds.save(objs={'dev_naive_bayes':gnb},
                  name="ds_dev", append=True)
```

```
[ ]: oml.ds.dir()
```

```
[ ]: sorted(oml.ds.load(name="ds_dev"))
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

Otra posibilidad (Opción 2) es hacer un poco de refactorización del código, encapsularlo en la función dev_gnb_train que se almacena en el repositorio de scripts de la base de datos y posteriormente se ejecuta con la funcionalidad oml.table_apply, que permite inyectar una tabla como dato den entrada a una función.
De esta manera se guardan en la base de datos tanto el código de creación del modelo, como el modelo en sí mismo (para almacenar el modelo se usa el Data Store al igual que en la Opción 1).
En esta Opción 2 no se extraen los datos de la base de datos, la ejecución completa ocurre en el servidor de la base de datos, aunque el control se tiene desde Jupyter.
Una vez construido el modelo, se puede traer a Jupyter para su inspección. El modelo creado es exactamente el mismo que en la Opción 1.
Esta forma de trabajo es lo que se denomina Ejecución Embebida.

OPTION 2: Embedded execution in Oracle database method

Model can be created in the database using python without pulling the data out.

Create a function for embedded execution to do the same training.

```
[ ]: def dev_gnb_train(dat):
         import oml
         import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.naive_bayes import GaussianNB


         X=dat[['DISTANCE', 'TRIP_DURATION',  'TMED', 'PREC',  'WEEKDAY',
         y=dat[['PRED']]
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz
         gnb = GaussianNB()
         gnb.fit(X_train, y_train)
         oml.ds.save(objs={'dev_naive_bayes2':gnb},
                     name="ds_dev", append=True)
         return gnb
```

```
[ ]: oml.script.create("dev_gnb_train", dev_gnb_train, overwrite = True)
```

```
[ ]: res = oml.table_apply(data=ORDERS_ML, func='dev_gnb_train',oml_connect
     res
```

Ejecución embebida del modelo predictivo desde Python

La ejecución del modelo puede hacerse de forma normal en Python, ya que es un modelo creado con Sklearn.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

Pero en el notebook se realiza una ejecución embebida, donde es la base de datos la que se encarga de hacerlo. Para ello, al igual que en el caso anterior de la Opción 2, se crea la función `dev_gnb_predict` y se almacena en el repositorio de scripts de la base de datos.

Create a function to predict a new observation, to be executed in embedded mode.

Any of the two models created in option 1 and 2 can be used. This means the database can take models created using the traditional way.

```
[ ]: def dev_gnb_predict(dist,trdu, tmed, prec,weday,ite):
       import pandas as pd
       import oml
       from sklearn.naive_bayes import GaussianNB

       objs=oml.ds.load(name="ds_dev", to_globals=False)

       df = pd.DataFrame({'DISTANCE': dist, 'TRIP_DURATION': trdu, 'TMED': tmed,'
       pred = objs["dev_naive_bayes2"].predict(df[['DISTANCE', 'TRIP_DURATION', '
       return  pd.DataFrame(pred, columns=["PRED"])
```

```
[ ]: oml.script.create("dev_gnb_predict", dev_gnb_predict, overwrite = True)
```

```
[ ]: oml.script.dir()
```

Para invocar de forma embebida esta función con unos parámetros cualesquiera, se usa el método `oml.do_eval` como se muestra a continuación:

```
[ ]: res = oml.do_eval(func='dev_gnb_predict',dist=12,trdu=33.2, tmed=3.3, prec=0,weday=2,ite=1,oml_connect=True)
     res
```

Ejecución del modelo predictivo desde SQL

El código de las funciones se encuentra almacenado en el repositorio de scripts de la base de datos y el objeto que almacena el modelo también está en el Data Store.
Estos hechos, junto con la funcionalidad de OML4Py que añade nuevos verbos en el lenguaje SQL, permite ejecutar código Python desde una sentencia SQL.

La sentencia que se muestra a continuación permite hacer la misma predicción que en el ejemplo anterior de Python. En SQL `pyqEval` equivale a `oml.do_eval` en Python.
Esta sentencia se ejecuta conectando con SQLcl, SQLPlus o SQLDeveloper con el usuario SOE a la PDB APP_ROOT.

El primer parámetro son los datos de entrada, en forma de un documento JSON. El segundo especifica el nombre de las columnas de salida y su tipo de dato. El tercero es el nombre de la función que se quiere invocar, que previamente ha tenido que ser almacenada en la base de datos, tal y como se hizo en las secciones anteriores.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

Execution from database (without Jupyter or python)

SQL sentence to run the same prediction operation from a regular database connection.

```
SELECT *
FROM table(pyqEval(
'{"dist": 12, "trdu": 33.2, "tmed": 3.3, "prec": 0, "weday":2, "ite":1,"oml_connect":1}',
'{"PRED":"number"}',
'dev_gnb_predict'));
```

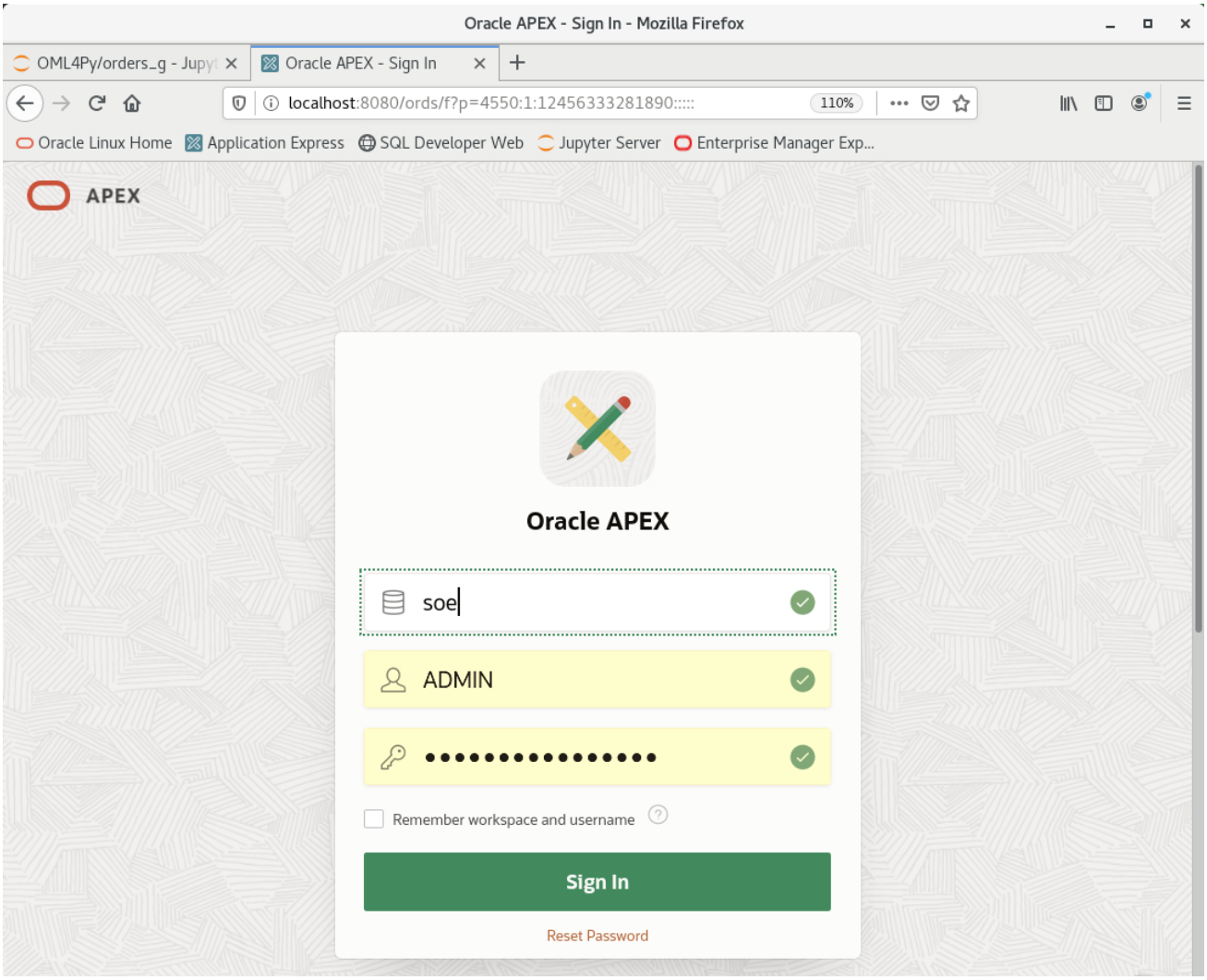
```
❯ sql soe/soe@localhost:1521/app_root


SQLcl: Release 21.4 Production on Thu Jun 30 10:53:21 2022

Copyright (c) 1982, 2022, Oracle.  All rights reserved.

Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

login.sql found in the CWD. DB access is restricted for login.sql.
Adjust the SQLPATH to include the path to enable full functionality.
SQL> SELECT *
    FROM table(pyqEval(
    '{"dist": 12, "trdu": 33.2, "tmed": 3.3, "prec": 0, "weday":2, "ite":1,
"oml_connect":1}',
   '{"PRED":"number"}',
   'dev_gnb_predict'));

   PRED
_______
      0

SQL> exit
```

Exposición y ejecución del modelo predictivo desde REST

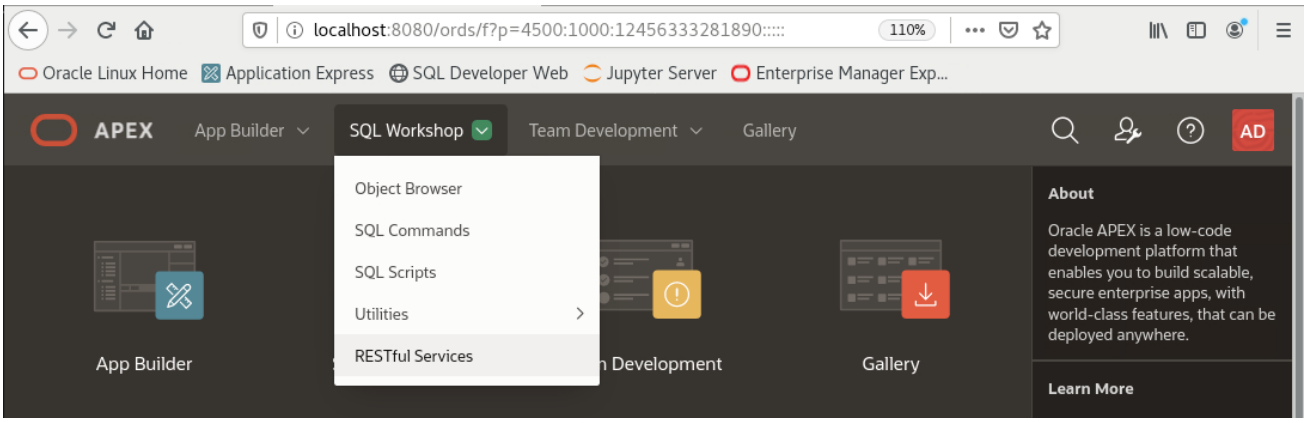
En último lugar se va a exponer la función python `dev_gnb_predict` que permite ejecutar el modelo predictivo a través de ORDS. Esto permitirá su invocación desde una petición HTTP POST.

Por comodidad este proceso se va a realizar desde APEX. En primer lugar se accede a la página y se accede con el usuario SOE.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

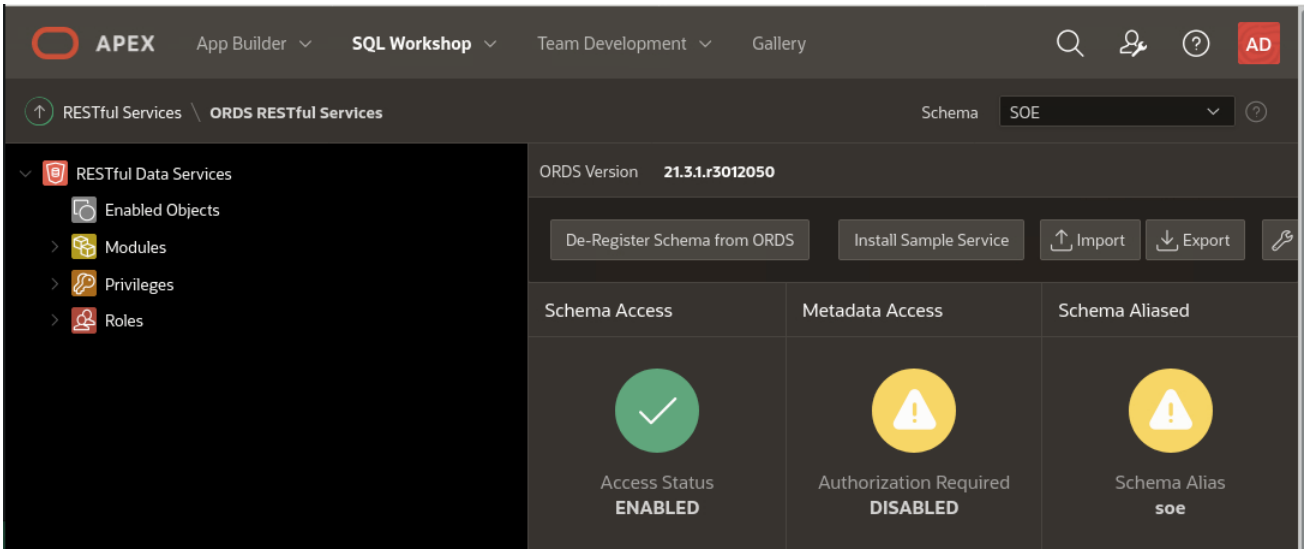


En el menú superior, despliegue SQL Workshop y RESTful Services

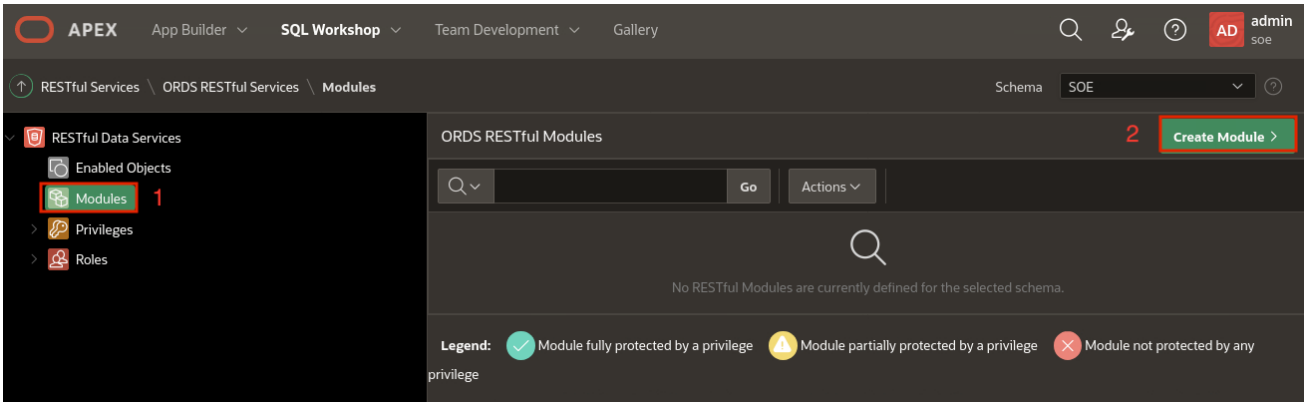


Esto despliega el menú de servicios REST que se ofrecen a través de ORDS.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

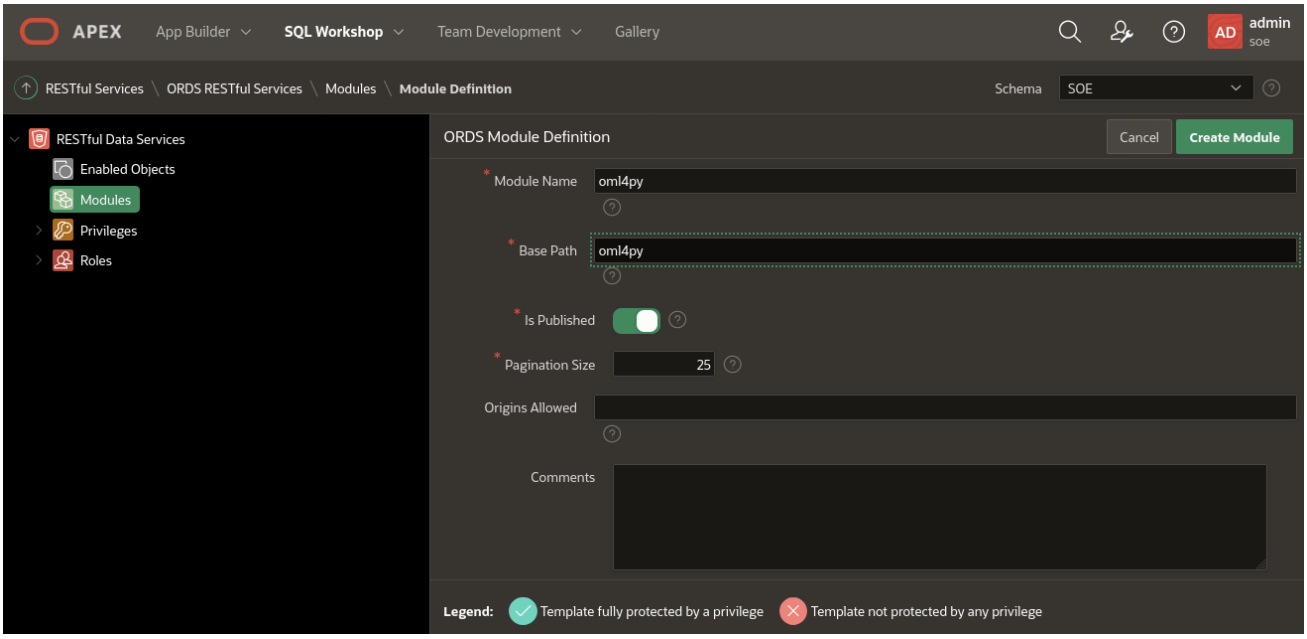


Se crea un módulo nuevo, en el menú de la izquierda haga click en Modules (1) y después en la derecha en Create Module (2).

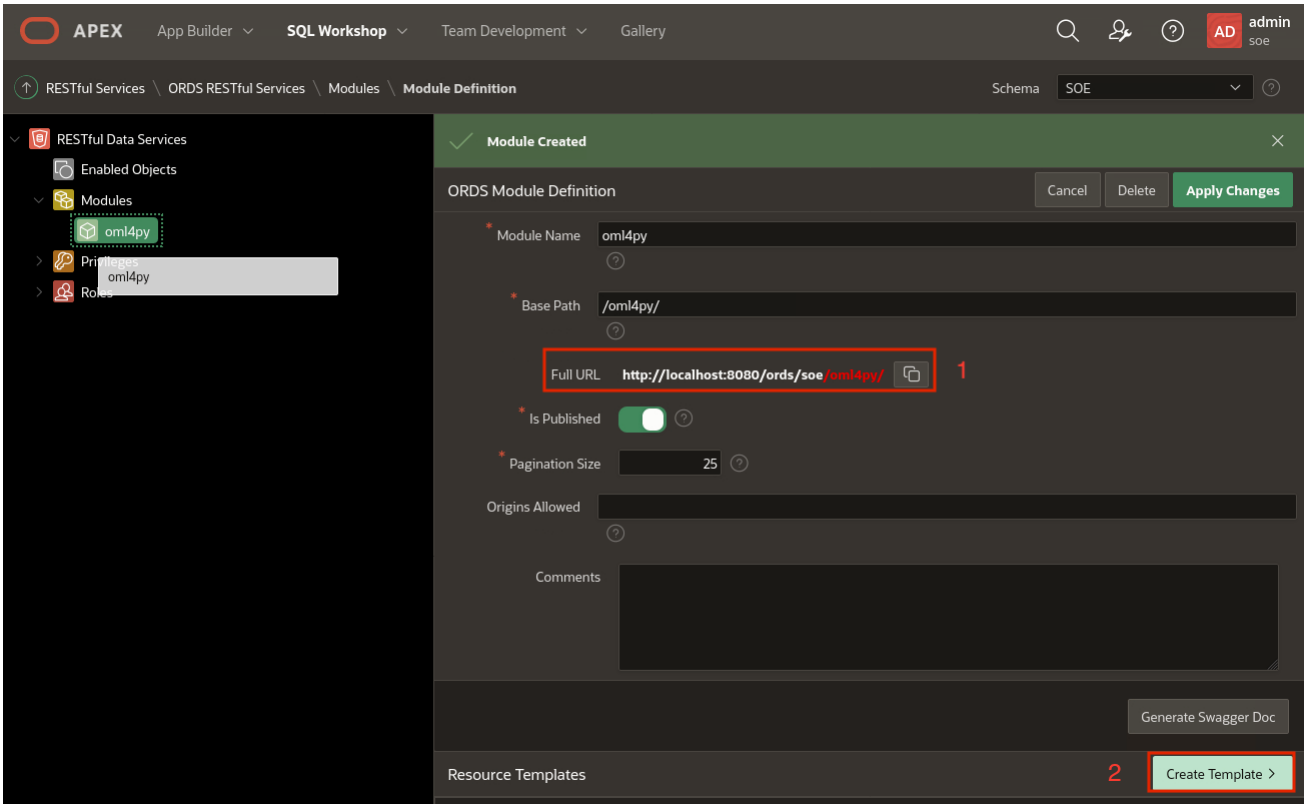


En la nueva pantalla, se le da un nombre al módulo y un base path con el que se construirá la URL para hacer la petición.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

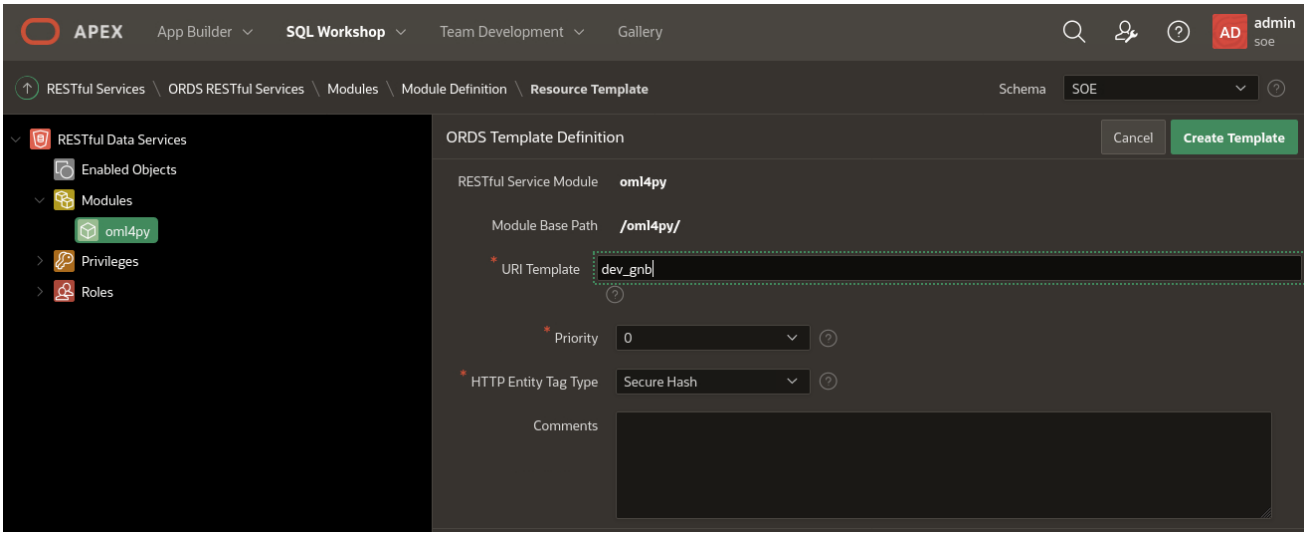


Una vez creado, se muestra la url o el endpoint base donde se publicará (1). Se pulsa el botón Create Template(2).

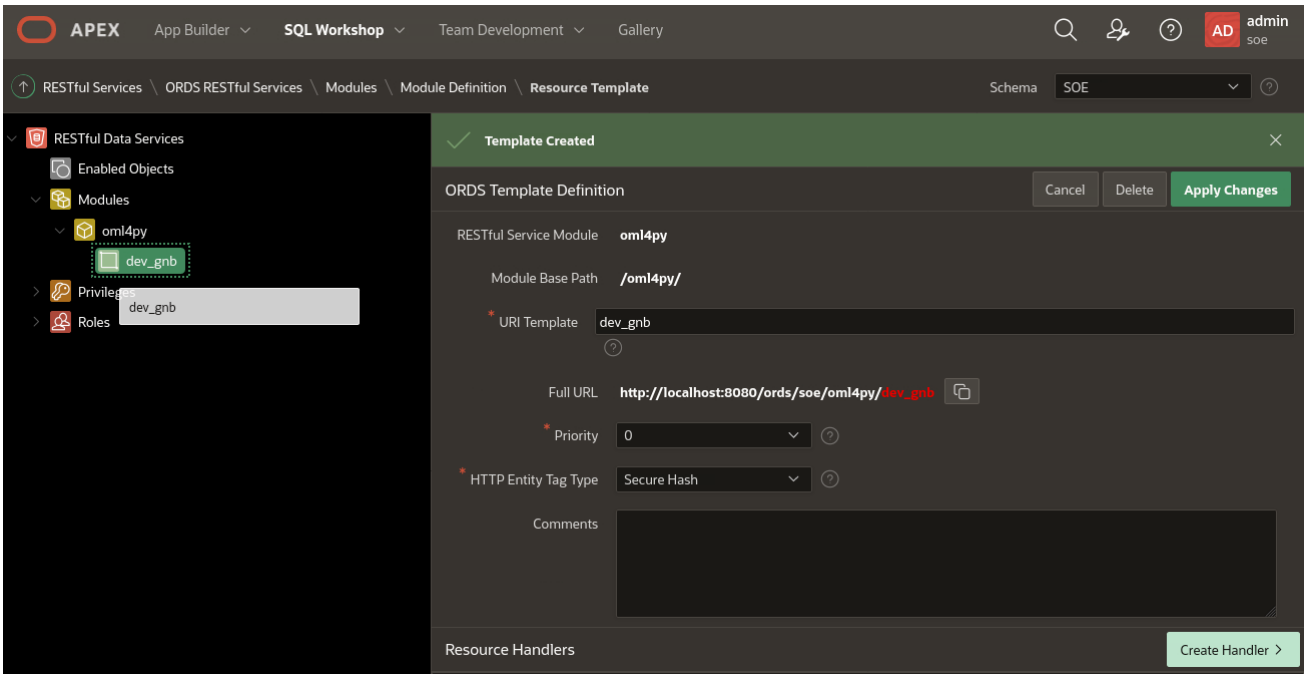


Se le da un subpath dentro del módulo al Template. A este Template se le asocian los métodos HTTP (GET, POST, PUT, DELETE) y las acciones que desencadenarán. Una vez introducida la URI, se hace click en Create Template.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

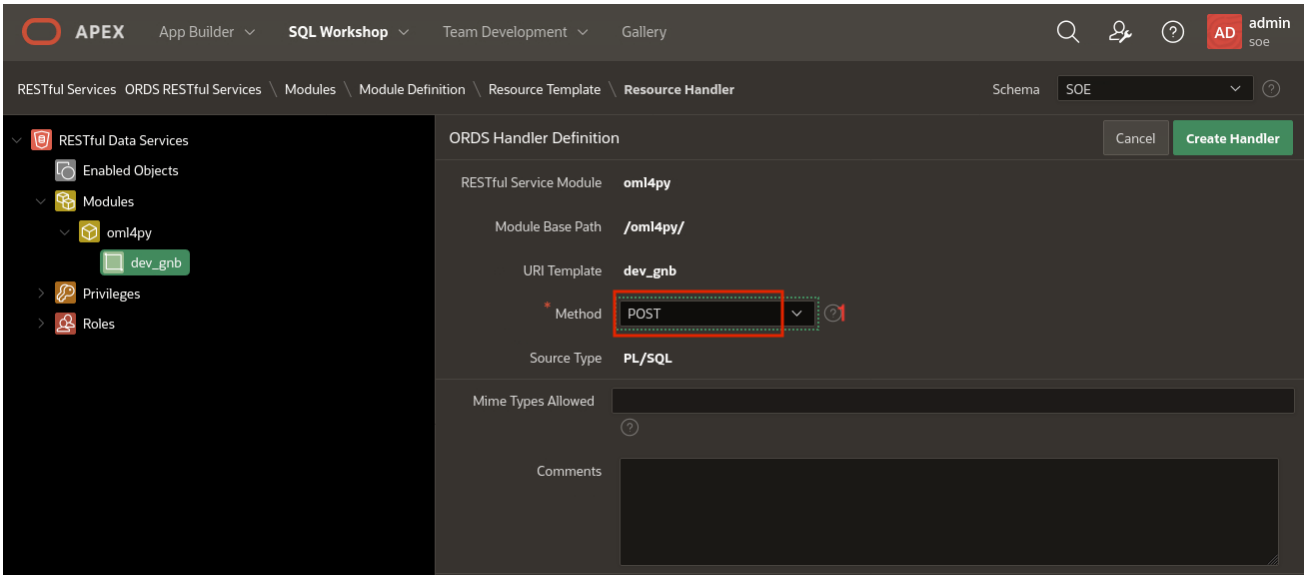


De nuevo se muestra un resumen, con la Full URL de acceso. Ahora se está en disposición de crear un Handler para los diferentes métodos HTTP.



Se selecciona POST en Method(1) y automáticamente el Source Type cambia a PL/SQL.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.



Más abajo, en el panel Source hay que introducir el código PL/SQL para invocar la sentencia SQL que invocará la función Python que usa el modelo GNB (Gaussian Naive Bayes) para calcular la predicción.

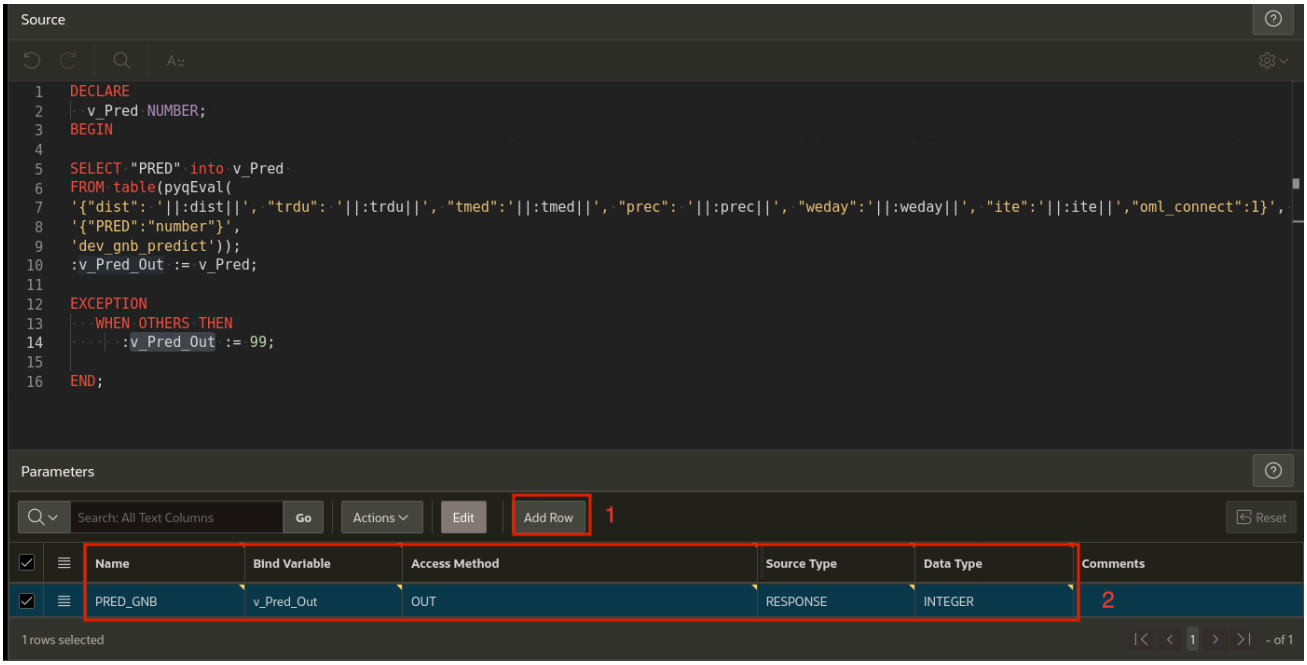
El código es el siguiente:

```
DECLARE
  v_Pred NUMBER;
BEGIN

SELECT "PRED" into v_Pred
FROM table(pyqEval(
```

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

```
'{"dist": '||:dist||', "trdu": '||:trdu||', "tmed":'||:tmed||', "prec": '||:prec||',
"weday":'||:weday||', "ite":'||:ite||',"oml_connect":1}',
'{"PRED":"number"}',
'dev_gnb_predict'));
:v_Pred_Out := v_Pred;

EXCEPTION
   WHEN OTHERS THEN
      :v_Pred_Out := 99;

END;
```

Las variables que vienen en la petición HTTP POST se prefijan con ':', de forma que se inyectarán en el bloque PL/SQL antes de ejecutarse.
Se define una variable `:v_Pred_Out` que se usará para devolver el resultado en la respuesta.
Esta parte se configura en la sección Parameters del fondo de la página. En primer lugar se hace click en el botón Add Row (1) y después se rellena el nombre del parámetro en la respuesta (PRED_GNB), la variable a la que se asocia(v_Pred_Out), el método de acceso (OUT), el Source Type(RESPONSE) y el Data Type(INTEGER).



Para completar, en la parte superior click en Create Handler.

Con estos pasos está publicada la función `dev_gnb_predict` en http://localhost:8080/ords/soe/oml4py/dev_gnb mediante una petición POST que incluya las variables necesarias: dist, trdu, tmed, prec, weday e ite.

Por simplicidad, tanto el cifrado como la autenticación están desactivados para hacer el proceso más sencillo, pero ambas opciones se pueden activar para proteger debidamente el endpoint REST.

Para realizar una llamada a ORDS, se puede usar la línea de comandos y curl, que permite realizar una petición HTTP POST.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

La petición debe llevar una cabecera de content-type y los datos necesarios para parametrizar la llamada a la función Python, las 5 variables mencionadas: dist, trdu, tmed, prec, weday e ite. Para ello se crea un documento JSON con ellas.

La petición es como se muestra a continuación:

```
curl --header "Content-type: application/json" \
--data '{ "dist":12, "trdu":33.2, "tmed":3.3, "prec":0, "weday":2, "ite":1}'  \
http://localhost:8080/ords/soe/oml4py/dev_gnb
```

Al ejecutarla, la respuesta se recibe en forma de documento JSON con el resultado etiquetado tal y como se configuró en el último paso de la configuración del Handler, con el nombre PRED_GNB.

```
❯ curl --header "Content-type: application/json" --data '{ "dist":12, "trdu":33.2, "tmed":3.3, "prec
":0, "weday":2, "ite":1}' http://localhost:8080/ords/soe/oml4py/dev_gnb
{"PRED_GNB":0}%
0.16  18G  14:37:34
```

Para más información de OML4Py: https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/index.html

Para más información de ORDS: https://www.oracle.com/database/technologies/appdev/rest.html

¡Enhorabuena! ha completado las actividades de esta sección y también todas las de la jornada.

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.