CodeXChange - PowerBuilder - Google Maps Distance API Example

Strong attribution to Daryl Foster:

https://community.appeon.com/index.php/qna/q-a/profile/2164-daryl-foster

Problem:

A client in the transportation space needed a way to add garage to garage time to hourly trips. So if you book a car service for 4.0 hours and the nearest base is 30 minutes away they wanted to add 60 minutes (30 minutes each way) to the trip in order to price it the way they want.

Approach:

Client and Developer agreed on a list of 30 cities in their country that had 'bases' or depots where cars could be counted on to go to/from the pickup location, perform the service and go back to the depot.

1. Create city_centers table with city_centers.csv file provided.*
2. Create place table with place.csv file provided.*
3. Create minutes_to_depot table as described.
4. The PowerBuilder project folder with all of the necessary objects is in geopostcodes - CXC.zip. The only change I made was to truncate the Google Maps API Key.

*You may need to delete the geography column before you import it as the data is self-generating.

These examples use PowerBuilder 2021 and Microsoft SQL Server 2019.

Basic Data:

PLACES WITH DEPOTS:

Table Name: **city_centers**

A list we found on the internet and culled based on our knowledge of where car service suppliers actually exist that includes latitude and longitude. 30 rows.

```
USE [Cogenten_GEO]
GO

/****** Object:  Table [dbo].[city_centers]    Script Date: 11/9/2021 9:35:06 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
```

```
GO

CREATE TABLE [dbo].[city_centers](
	[ID] [int] NOT NULL,
	[city] [char](80) NULL,
	[latitude] [float] NULL,
	[longitude] [float] NULL,
	[Country] [char](50) NULL,
	[ISO] [char](2) NULL,
	[admin_name] [char](50) NULL,
	[capital] [char](50) NULL,
	[population] [float] NULL,
	[geography]  AS
([geography]::STGeomFromText((((('POINT('+CONVERT([varchar](20),[longitude]))+'
')+CONVERT([varchar](20),[latitude]))+')',(4326))),
	[country_code] [char](2) NULL,
PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

PLACES IN GERMANY: POSTAL CODES AND AIRPORTS:

Table Name: **place**

A combination of airport data and postal code data found on the internet along with latitude and longitude. ~8,400 rows.

```
USE [Cogenten_GEO]
GO

/****** Object:  Table [dbo].[place]    Script Date: 11/9/2021 9:33:36 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[place](
	[postal_code] [char](10) NOT NULL,
	[place_name] [varchar](80) NULL,
	[state] [varchar](80) NULL,
	[state_abbr] [varchar](30) NULL,
	[city] [varchar](50) NULL,
	[latitude] [float] NULL,
	[longitude] [float] NULL,
	[geography]  AS
([geography]::STGeomFromText((((('POINT('+CONVERT([varchar](20),[longitude]))+'
')+CONVERT([varchar](20),[latitude]))+')',(4326))),
	[country_code] [char](2) NULL,
 CONSTRAINT [AK_KEY_1_PLACE] UNIQUE NONCLUSTERED
(
	[postal_code] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

JOINED DATA VIA SPATIAL QUERY USING SQL SERVER STORED PROCEDURE:

We used a stored procedure to find the closest distance (as the crow flies) from each postal code in German to each of the City Centers that met our criteria.

Stored Procedure: dbo.get_closest_city_center:

Here is the code to create the spatial stored procedure…

```
USE [Cogenten_GEO]
GO

/****** Object:  StoredProcedure [dbo].[get_closest_city_center]    Script Date:
11/8/2021 7:39:28 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[get_closest_city_center]
@long float,
@lat float,
@country_code char(2) output,
@city char(50) output,
@latitude float output,
@longitude float output,
@distance float output
AS
DECLARE @me geography = 'POINT(' + CAST(@long AS VARCHAR(20)) + ' ' + CAST(@lat AS
VARCHAR(20)) + ' 4326)';
SELECT TOP(1) country_code, city, latitude, longitude,
round(geography.STDistance(@me),2)/1000 AS distance_meters
FROM city_centers
ORDER BY distance_meters
GO
```

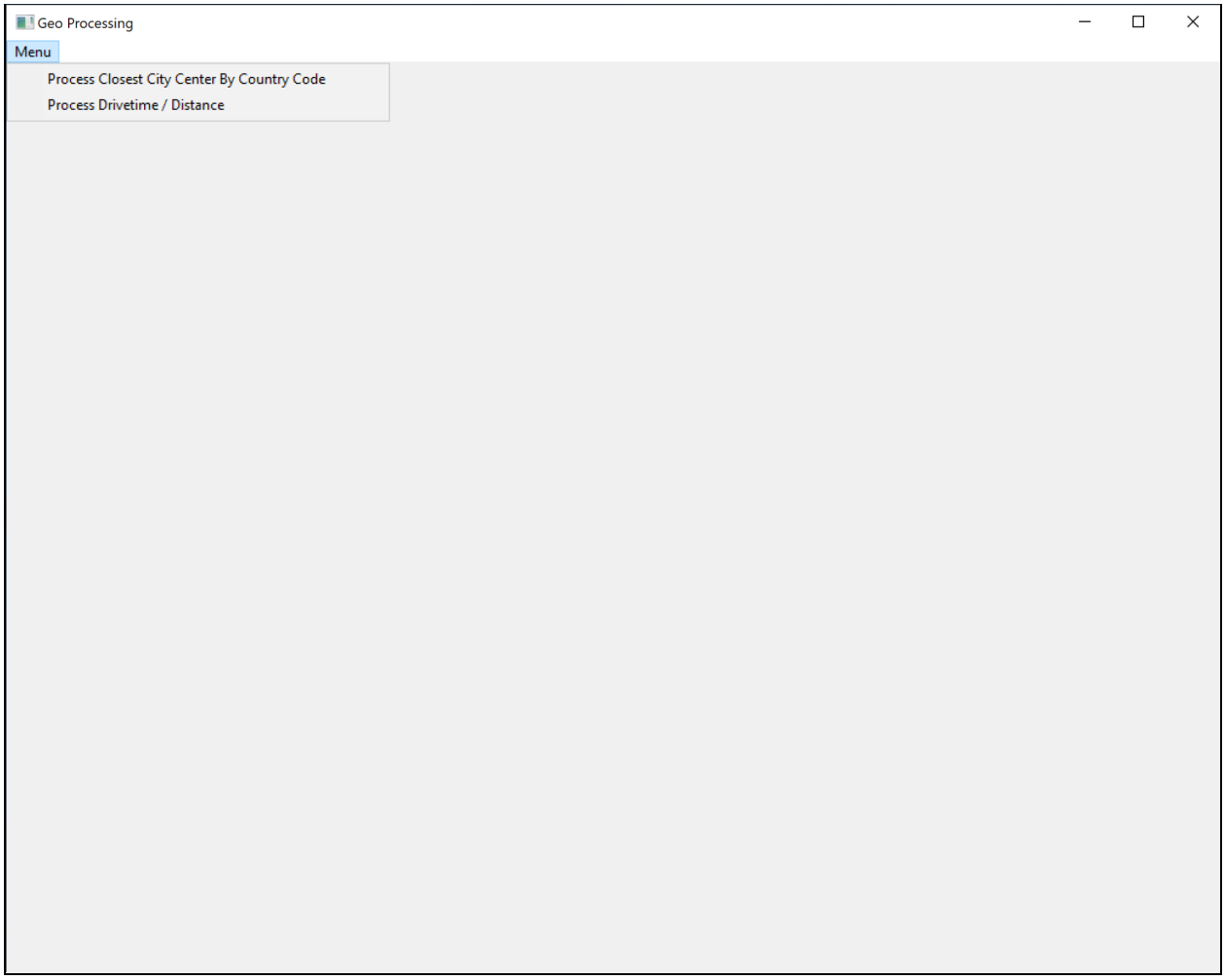
RESULTING TABLE IN THIS TABLE: minutes_to_depot

```
USE [Cogenten_GEO]
GO

/****** Object:  Table [dbo].[minutes_to_depot]    Script Date: 11/9/2021 9:32:04 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
```

```
GO

CREATE TABLE [dbo].[minutes_to_depot](
	[ID] [bigint] IDENTITY(1,1) NOT NULL,
	[country_code] [varchar](2) NOT NULL,
	[city_airport] [varchar](50) NULL,
	[postal_code] [varchar](10) NULL,
	[place_latitude] [decimal](10, 6) NULL,
	[place_longitude] [decimal](10, 6) NULL,
	[city_center] [varchar](80) NULL,
	[d_latitude] [decimal](10, 6) NULL,
	[d_longitude] [decimal](10, 6) NULL,
	[distance_crow] [decimal](9, 2) NULL,
	[distance_gm_km] [decimal](9, 2) NULL,
	[drivetime] [int] NULL,
 CONSTRAINT [MTD_pkey] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

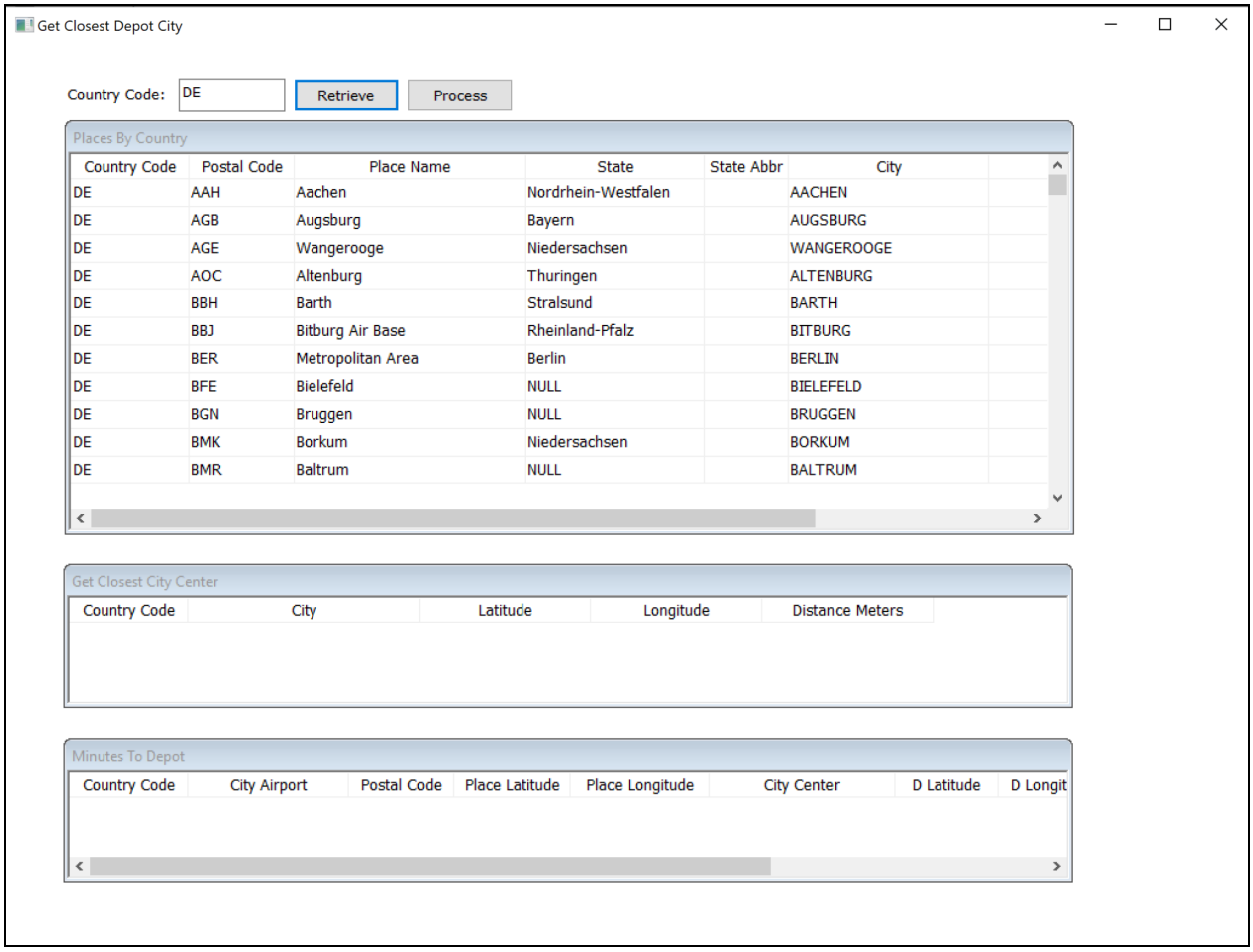
PowerBuilder Application Screenshots:

There are two windows with various objects for processing.



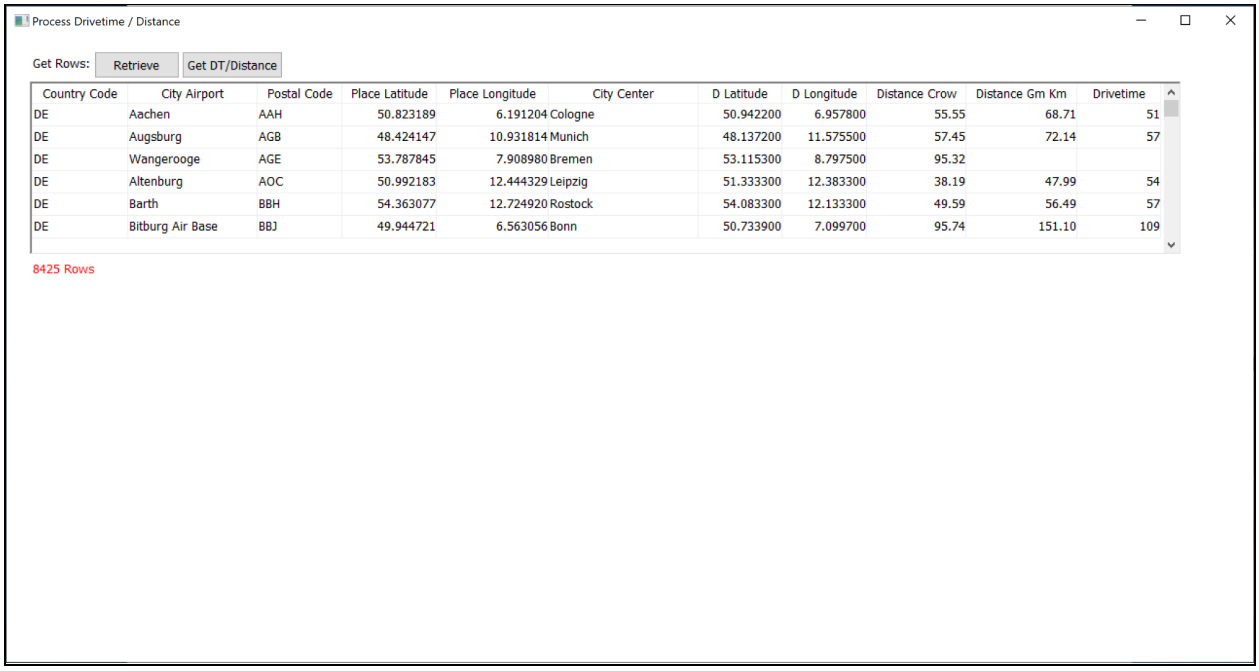
Get Closest Depot City:

1. The first datawindow retrieves the ~8,400 places from the place table.
2. Clicking the ‘Process’ button causes the application to iterate through each row in the first datawindow and call a stored procedure which gets the closest depot (or city center) from the city_center table along with ‘as the crow flies’ distance in kilometers and inserts that data into the second data window ‘Get Closest City Center’.
3. The data from the ‘Get Closest City Center’ is combined with the data from ‘Places By Country’ and then inserted into ‘minutes_to_depot’ table.

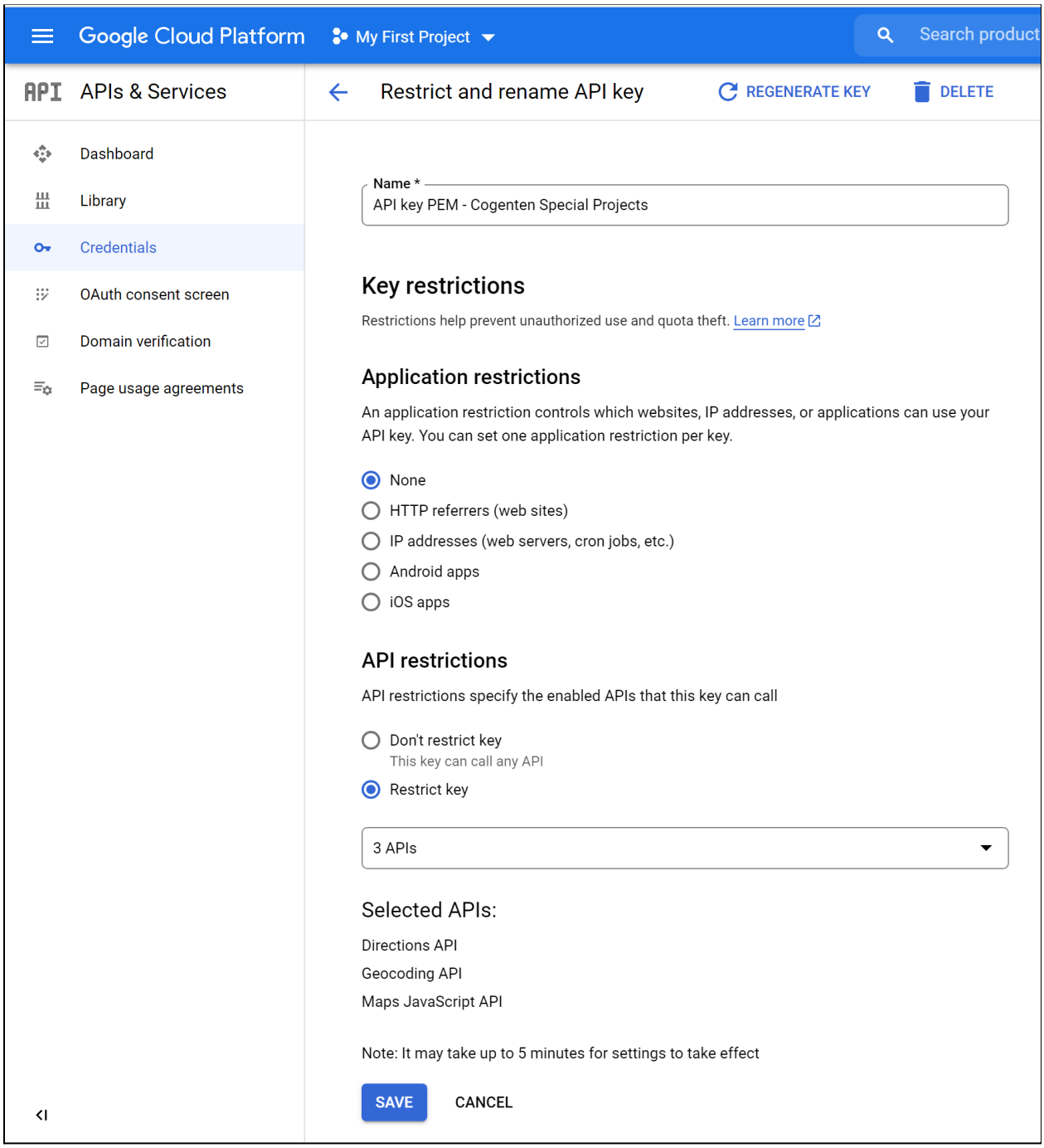


Process Drivetime / Distance:

1. This window has a datawindow with all rows from the ‘minutes_to_depot’ table.
2. Clicking the ‘Get DT/Distance’ button causes the application to iterate through each row, get the pickup and dropoff latitude/longitudes and call function ‘f_call_gm_directions_api’ and the resulting JSON is stored in a variable.
3. Drivetime and distance VALUES are then extracted/parsed from the JSON.
4. The same row is populated with these values in datawindow columns ‘distance_gm_km’ and ‘drivetime’ the datawindow is updated.


Process Drivetime / Distance
Get Rows:
Retrieve
Get DT/Distance
Country Code City Airport Postal Code Place Latitude Place Longitude City Center D Latitude D Longitude Distance Crow Distance Gm Km Drivetime
DE Aachen AAH 50.823189 6.191204 Cologne 50.942200 6.957800 55.55 68.71 51
DE Augsburg AGB 48.424147 10.931814 Munich 48.137200 11.575500 57.45 72.14 57
DE Wangerooge AGE 53.787845 7.908980 Bremen 53.115300 8.797500 95.32
DE Altenburg AOC 50.992183 12.444329 Leipzig 51.333300 12.383300 38.19 47.99 54
DE Barth BBH 54.363077 12.724920 Rostock 54.083300 12.133300 49.59 56.49 57
DE Bitburg Air Base BBJ 49.944721 6.563056 Bonn 50.733900 7.099700 95.74 151.10 109
8425 Rows

Even if you have a Google Maps API key, you still need to ensure that the ‘Directions API’ is selected for the key as shown below.



My thanks to Daryl Foster who helped immensely with the Google Maps calls and Chris Pollach who helped me figure out that I needed a stored procedure to get closest depot / city_center.