

## **Calculating distance between zip codes: A database case study in consuming web services**

Mohammad Dadashzadeh  
Oakland University

### **ABSTRACT**

Travel distance is a variable in predicting consumer behavior in choosing a service location. In predictive modeling applications in medicine, there is often a need to calculate the distance of a patient from one or more healthcare facilities. With de-identified patient data, a patient's zip code becomes the basis for calculating distance. Therefore, the problem of calculating distance between zip codes is a recurring data pre-processing step in such applications. This paper presents the development of a case study for the database course to automate a solution to the problem while providing teaching opportunities in Structured Query Language (SQL), Microsoft Access Visual Basic for Applications (VBA) programming, and consuming web services.

Keywords: IS Curriculum, Database Course, Location Analytics, Travel Distance Calculation, Geocoding API, Map Web Services

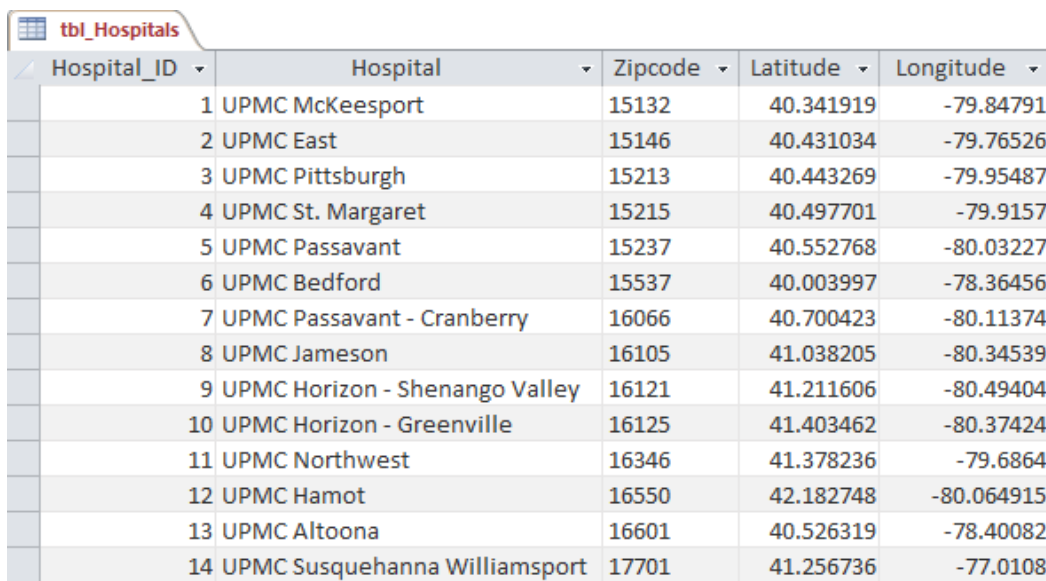


Copyright statement: Authors retain the copyright to the manuscripts published in AABRI journals. Please see the AABRI Copyright Policy at <http://www.aabri.com/copyright.html>

## INTRODUCTION

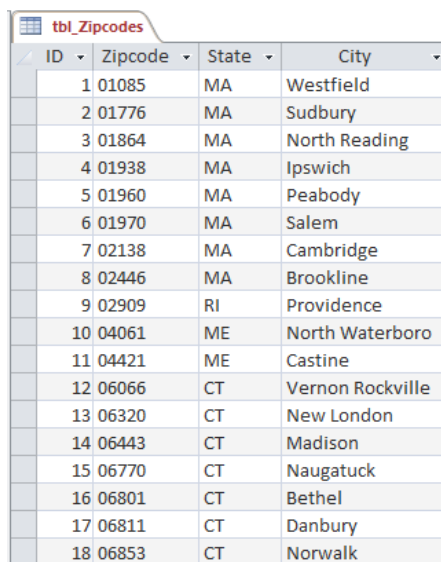
Travel distance is a variable in formulating an understanding of why a specific service location amongst several competing ones is chosen by a consumer. The need to calculate geographic distance occurs in various contexts including tourism research (Nyaupane, Graefe, & Burns, 2003), emergency medical services (EMS) system planning (Hsia et al., 2017), and healthcare provision (Tsai, Orav, & Jha, 2015).

To fix ideas, let us consider an example of calculating distance between patients and hospitals. The data structure in Microsoft Access consists of two tables: *tbl\_Hospitals* (Figure 1) that records the zip code of 14 service locations of interest, and *tbl\_Zipcodes* (Figure 2) that captures the 1,505 distinct patient zip codes for which the distance to each service location needs to be calculated.



Hospital_ID	Hospital	Zipcode	Latitude	Longitude
1	UPMC McKeesport	15132	40.341919	-79.84791
2	UPMC East	15146	40.431034	-79.76526
3	UPMC Pittsburgh	15213	40.443269	-79.95487
4	UPMC St. Margaret	15215	40.497701	-79.9157
5	UPMC Passavant	15237	40.552768	-80.03227
6	UPMC Bedford	15537	40.003997	-78.36456
7	UPMC Passavant - Cranberry	16066	40.700423	-80.11374
8	UPMC Jameson	16105	41.038205	-80.34539
9	UPMC Horizon - Shenango Valley	16121	41.211606	-80.49404
10	UPMC Horizon - Greenville	16125	41.403462	-80.37424
11	UPMC Northwest	16346	41.378236	-79.6864
12	UPMC Hamot	16550	42.182748	-80.064915
13	UPMC Altoona	16601	40.526319	-78.40082
14	UPMC Susquehanna Williamsport	17701	41.256736	-77.0108

**Figure 1.** Microsoft Access Table Listing Various Service Locations



ID	Zipcode	State	City
1	01085	MA	Westfield
2	01776	MA	Sudbury
3	01864	MA	North Reading
4	01938	MA	Ipswich
5	01960	MA	Peabody
6	01970	MA	Salem
7	02138	MA	Cambridge
8	02446	MA	Brookline
9	02909	RI	Providence
10	04061	ME	North Waterboro
11	04421	ME	Castine
12	06066	CT	Vernon Rockville
13	06320	CT	New London
14	06443	CT	Madison
15	06770	CT	Naugatuck
16	06801	CT	Bethel
17	06811	CT	Danbury
18	06853	CT	Norwalk

**Figure 2.** Microsoft Access Table Showing 18 of 1,505 Distinct Patient Zip Codes

Given a user-defined function named *fnDistanceBetweenZipcodes* (Figure 3) to calculate and return the distance between two zip codes, the following SQL statement would create the table *tbl\_ZipcodeDistances* (Figure 4) consisting of 21,070 (i.e., 1505 x 14) rows showing the desired distances:

```
SELECT      ID, tbl_Zipcodes.Zipcode AS Zip1, tbl_Hospitals.Hospital,
           tbl_Hospitals.Zipcode AS Zip2,
           fnDistanceBetweenZipcodes([Zip1], [Zip2]) AS Distance
INTO        tbl_ZipcodeDistances
FROM        tbl_Zipcodes, tbl_Hospitals
```

```
Function fnDistanceBetweenZipcodes(zipcode1 As String, zipcode2 As String) As Single
'Function to calculate distance between zip codes ...
'Returns 0 ...
fnDistanceBetweenZipcodes = 0
End Function
```

**Figure 3.** The Structure of VBA Function to Return Calculated Distance Between Zip Codes

ID	Zip1	Hospital	Zip2	Distance
1	01085	UPMC McKeesport	15132	0
1	01085	UPMC East	15146	0
1	01085	UPMC Pittsburgh	15213	0
1	01085	UPMC St. Margaret	15215	0
1	01085	UPMC Passavant	15237	0
1	01085	UPMC Bedford	15537	0
1	01085	UPMC Passavant - Cranberry	16066	0
1	01085	UPMC Jameson	16105	0
1	01085	UPMC Horizon - Shenango Valley	16121	0
1	01085	UPMC Horizon - Greenville	16125	0
1	01085	UPMC Northwest	16346	0
1	01085	UPMC Hamot	16550	0
1	01085	UPMC Altoona	16601	0
1	01085	UPMC Susquehanna Williamsport	17701	0
2	01776	UPMC McKeesport	15132	0
2	01776	UPMC East	15146	0
2	01776	UPMC Pittsburgh	15213	0
2	01776	UPMC St. Margaret	15215	0
2	01776	UPMC Passavant	15237	0
2	01776	UPMC Bedford	15537	0
2	01776	UPMC Passavant - Cranberry	16066	0
2	01776	UPMC Jameson	16105	0

**Figure 4.** Sample of the Output Table *tbl\_ZipcodeDistances* Showing Returned Distances

The final solution step would be to create a cross tabulation of data in the table, *tbl\_ZipcodeDistances*, that more clearly shows the distance between each distinct patient zip code and the 14 service location zip codes (Figure 5). This can be accomplished using the Crosstab Query Wizard in Microsoft Access as:

```

TRANSFORM      First(tbl_ZipcodeDistances.[Distance]) AS FirstOfDistance
SELECT
FROM           tbl_ZipcodeDistances
GROUP BY      tbl_ZipcodeDistances.[Zip1]
PIVOT         tbl_ZipcodeDistances.[Zip2]

```

Zip1	15132	15146	15213	15215	15237	15537	16066	16105	16121	16125	16346	16550	16601	17701
01085	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01776	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01864	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01938	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01960	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01970	0	0	0	0	0	0	0	0	0	0	0	0	0	0
02138	0	0	0	0	0	0	0	0	0	0	0	0	0	0
02446	0	0	0	0	0	0	0	0	0	0	0	0	0	0
02909	0	0	0	0	0	0	0	0	0	0	0	0	0	0
04061	0	0	0	0	0	0	0	0	0	0	0	0	0	0
04421	0	0	0	0	0	0	0	0	0	0	0	0	0	0
06066	0	0	0	0	0	0	0	0	0	0	0	0	0	0
06320	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 5.** Layout of the Desired Final Output

As it has been outlined above, the software architecture of a solution to the problem builds upon concepts, namely SQL and VBA programming, taught in a typical database course in the MIS curriculum using Microsoft Access. However, the important new teaching opportunity that emerges is in completing the function *fnDistanceBetweenZipcodes* to return the correct distance by utilizing existing web services.

## DISTANCE CALCULATION USING ZIP CODE WEB SERVICES

Zip-Codes.com (2019) is a supplier of US and Canadian zip code database/directory. It also provides a subscription-based Application Programming Interface (API) to its web services for functions including obtaining the zip code of an address, distance calculations, and radius searching. The Zip-Codes.com's API is a RESTful service (Richardson & Ruby, 2007) in that it accepts HTTP requests and provides responses in XML or JavaScript Object Notation (JSON) text formats. For example, the request to calculate the distance between zip codes **32504** and **90210** and receive the response in XML format can be sent via the following URL:

<https://api.zip-codes.com/ZipCodesAPI.svc/1.0/XML/CalculateDistance/ByZip?fromzipcode=32504&tozipcode=90210&key=DEMOAPIKEY>

where, the API subscription key of DEMOAPIKEY is used. The XML response from the web service would include text:

<DistanceInMiles>1835.096876886239</DistanceInMiles>

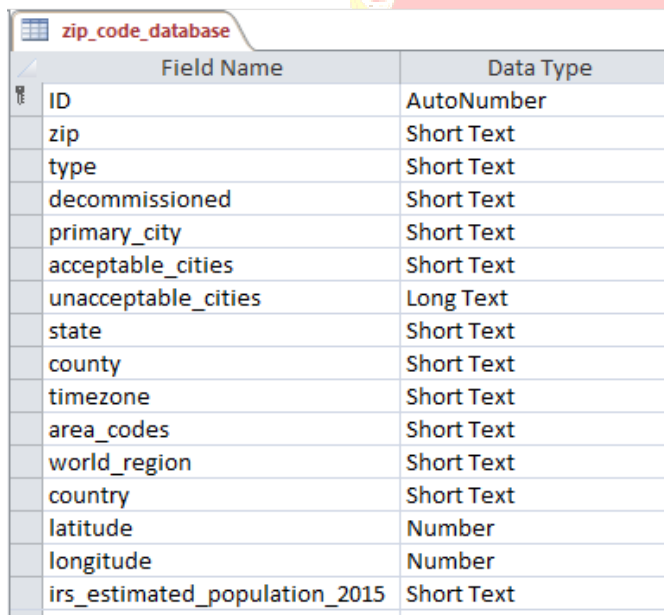
that can be parsed to obtain the calculated *straight-line* distance of 1835 miles.

The Appendix gives the completed code for the Microsoft Access user-defined function, *fnDistanceBetweenZipcodes*, that utilizes the Zip-Codes.com's API to calculate the needed distances.

## DISTANCE CALCULATION USING HAVERSINE FORMULA

The *straight-line* distance between two locations ignores the fact that there are no straight lines on a sphere such as Earth. The haversine formula (Wikipedia, 2019) can be used to calculate the shortest *spherical* distance between two points on the surface of a sphere, measured along the surface. Although the formula would not be completely accurate since the Earth is not a perfect sphere, it provides a good approximation for most applications. The basic formula utilizes latitude and longitude coordinates of the locations and can be expressed as a VBA function in Microsoft Access as shown in the Appendix.

This approach to calculating distances between zip codes requires that we obtain latitude and longitude coordinate values associated with each of our 1,505 distinct zip codes. Geocoding web services that take an address and return an actual or calculated latitude/longitude coordinate can provide that functionality. However, several sites including UnitedStatesZipCodes.org (2019) provide a "personal" version of their zip code database at no cost. Figure 6 shows the data structure of the zip code database as imported into Microsoft Access that supplies the needed latitude and longitude coordinates for distance calculation.



Field Name	Data Type
ID	AutoNumber
zip	Short Text
type	Short Text
decommissioned	Short Text
primary_city	Short Text
acceptable_cities	Short Text
unacceptable_cities	Long Text
state	Short Text
county	Short Text
timezone	Short Text
area_codes	Short Text
world_region	Short Text
country	Short Text
latitude	Number
longitude	Number
irs_estimated_population_2015	Short Text

**Figure 6.** Zip Code Database Supplying Latitude and Longitude Coordinates

## DISTANCE CALCULATION USING ROUTING WEB SERVICES

For applications requiring a more accurate calculation of travel distance between two locations, the solution may be found by using one of the many routing web services available including: Google Maps Platform's Directions API (Google, 2019), Bing Maps REST Services (Microsoft, 2019), ESRI's ArcGIS REST API (ESRI, 2019), and HERE Technologies' Routing API (HERE Technologies, 2019).

Using HERE Technologies' routing web service, the request to calculate the fastest travel route by car between zip codes **32504** (30.48°N, -87.19°E) and **90210** (34.10°N, -118.41°E) taking traffic conditions into account and departing immediately can be sent via the following URL:

```
https://route.api.here.com/routing/7.2/calculateroute.json
?waypoint0=30.48,-87.19&waypoint1=34.10,-118.41
&mode=fastest;car;traffic:enabled
&app_id=devportal-demo-20180625&app_code=9v2BkviRwi9Ot26kp2IysQ
&departure=now
```

where, the values specified for request parameters *app\_id* and *app\_code* are developer assigned subscription keys. The JSON response from the web service would include text:

```
"The trip takes <span class=\"length\">3369 km</span>
```

that can be parsed to obtain the calculated routing distance of 3369 kilometers (or 2093 miles).

The Appendix gives the completed code for the Microsoft Access function, *fnGetRoutingDistanceHERE\_API*, that utilizes HERE.com's API to calculate the needed distances.

## SUMMARY AND CONCLUSIONS

The IS 2010 Curriculum Guidelines for Undergraduate Degree Programs in Information Systems (ACM/AIS, 2010) recognizes the "emergence of a new architectural paradigm – service-oriented architecture, web services, software-as-a-service, and cloud computing" as a motivating factor for IS curriculum revision. As such, core skill sets for modern application development must include AJAX, XML, and web services in addition to the basic programming, web development, and database skills that have been taught for many years. Dadashzadeh (2010) describes a case study to expose IS students to web services as soon as they are introduced to the basics of HTML and programming. In this paper, the challenge of solving a real-world problem in data preparation for predictive modeling applications involving location analytics, that is, the problem of calculating distance between zip codes, has been presented as a pedagogical opportunity for the database course instructor to introduce students to modern web services and API's in a hands-on manner.



**REFERENCES**

- ACM/AIS. (2010). IS 2010 Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. New York, NY: Association for Computing Machinery.
- Dadashzadeh, M. (2010). Consuming Web Services: A Yahoo! Newsfeed Reader. *Journal of Information Systems Education*, 21(4), 355-360.
- ESRI. (2019). Routing and Direction with ArcGIS. Retrieved from <https://developers.arcgis.com/features/directions> on January 21, 2019.
- Google. (2019). The Directions API. Retrieved from <https://developers.google.com/maps/documentation/directions/start> on January 21, 2019.
- HERE Technologies. (2019). Routing API. Retrieved from <https://developer.here.com/documentation/routing/topics/request-a-simple-route.html> on January 21, 2019.
- Hsia, R.Y., Dai, M., Wei, R., Sabbagh, S., & Mann, N.C. (2017). Geographic Discordance Between Patient Residence and Incident Location in Emergency Medical Services Responses. *Annals of Emergency Medicine*, 69(1), 44-51.
- Microsoft. (2019). Bing Maps REST Services. Retrieved from <https://docs.microsoft.com/en-us/bingmaps/rest-services> on January 21, 2019.
- Nyaupane, G.P., Graefe, A., & Burns, R.C. (2003). Does Distance Matter? Differences in Characteristics, Behaviors, and Attitudes of Visitors Based on Travel Distance. In *Proceedings of the 2003 Northeastern Recreation Research Symposium*, 74-81. Newtown Square, PA: U.S. Department of Agriculture, Forest Service, Northeastern Research Station.
- Richardson, L., & Ruby, S. (2007). RESTful Web Services. Sebastopol, CA: O'Reilly Media.
- Tsai, T.C., Orav, E.J., & Jha, A.K. (2015). Care Fragmentation in the Postdischarge Period: Surgical Readmissions, Distance of Travel, and Postoperative Mortality. *JAMA Surgery*, 150(1), 59-64.
- UnitedStatesZipCodes.org. (2019). Zip Code Database. Retrieved from <https://www.unitedstateszipcodes.org/zip-code-database> on January 20, 2019.
- Wikipedia. (2019). Haversine Formula. Retrieved from [https://en.wikipedia.org/wiki/Haversine\\_formula](https://en.wikipedia.org/wiki/Haversine_formula) on January 20, 2019.
- Zip-Codes.com. (2019). Zip Code API. Retrieved from <https://www.zip-codes.com/zip-code-api.asp> on January 20, 2019.

**APPENDIX**

This appendix provides Microsoft Access VBA code referred to in the paper. This partial code for implementing the solution of calculating distance between zip codes is taught to, and shared with, the students along with the case study. A copy of the example database and the entire solution code is available from the author upon request.

**A. VBA function to calculate distance between zip codes using Zip-Codes.com API**

Function *fnDistanceBetweenZipcodes*(Zipcode1 As String, Zipcode2 As String) As Single

'Function to calculate distance between zip codes using Zip-Codes.com API ...

Dim strURL As String, BodyTxt As String

Dim Temp As Variant 'Return value ...

Dim strSearch As String

strSearch = "<DistanceInMiles>" 'Search pattern to look for in response ...

strURL = "http://api.zip-codes.com/ZipCodesAPI.svc/1.0/xml/CalculateDistance/ByZip?"

strURL = strURL & "fromzipcode=" & Zipcode1 & "&tozipcode=" & Zipcode2 & "&key=DEMOAPIKEY"

'Send the web service request ...

BodyTxt = *getResponse*(strURL)

'Parse the returned text for <DistanceInMiles> ...

If InStr(1, BodyTxt, strSearch, vbTextCompare) = 0 Then

    Temp = -1# '<DistanceInMiles> is absent in the response ... Return -1 ...

Else

    Temp = Mid(BodyTxt, InStr(1, BodyTxt, strSearch) + Len(strSearch))

    Temp = Mid(Temp, 1, InStr(1, Temp, "<") - 1)

    Temp = Val(Temp) 'Convert from string to numeric ...

End If

'Return Temp ...

fnDistanceBetweenZipcodes = Temp

End Function

Function *getResponse*(strURL As String) As String

'Submits the URL request and returns the response ...

Dim oXH As Object

Set oXH = CreateObject("msxml2.xmlhttp")

With oXH



```
.Open "get", strURL, False
.send
getResponse = .responseText
End With
```

```
Set oXH = Nothing
```

```
End Function
```

### **B. VBA function to calculate distance between geocoded locations using haversine formula**

Function *HaversineFormulaDistance*(Latitude1 As Single, Longitude1 As Single, Latitude2 As Single, Longitude2 As Single) As Single

```
Const EarthRadius = 3958 'in miles ...
```

```
Dim LatitudeDelta As Single, LongitudeDelta As Single
Dim A As Single, C As Single
```

```
'Convert from degrees to radians ...
```

```
LatitudeDelta = (Latitude2 * PI / 180) - (Latitude1 * PI / 180)
```

```
LongitudeDelta = (Longitude2 * PI / 180) - (Longitude1 * PI / 180)
```

```
A = ((Sin(LatitudeDelta / 2)) ^ 2) + Cos(Latitude1 * PI / 180) * Cos(Latitude2 * PI / 180) *
((Sin(LongitudeDelta / 2)) ^ 2)
```

```
C = 2 * ArcSin(Sqr(A))
```

```
HaversineFormulaDistance = EarthRadius * C
```

```
End Function
```

### **C. VBA function to calculate routing distance between geocoded locations using HERE.com**

Function *fnGetRoutingDistanceHERE\_API*(Latitude1 As Single, Longitude1 As Single, Latitude2 As Single, Longitude2 As Single) As Single

```
'Function to calculate routing distance between two geocodes using HERE.com API
```

```
Dim strURL As String, BodyTxt As String
```

```
Dim Temp As Variant 'Return value ...
```

```
Dim strSearch As String
```

```
'Search pattern to look for in the returned response ...
```

```
""The trip takes <span class=\"length\">
```

```
strSearch = "The trip takes " & Chr(60) & "span class=\""length\"" & Chr(62)
```

```
strURL = "https://route.api.here.com/routing/7.2/calculateroute.json?"  
strURL = strURL & "waypoint0=" & Trim(Str(Latitude1)) & "%2C" & Trim(Str(Longitude1))  
strURL = strURL & "&waypoint1=" & Trim(Str(Latitude2)) & "%2C" & Trim(Str(Longitude2))  
strURL = strURL & "&mode=fastest%3Bcar%3Btraffic%3Aenabled"  
strURL = strURL & "&app_id=devportal-demo-0180625&app_code=9v2BkviRwi9Ot26kp2IysQ"  
strURL = strURL & "&departure=now"
```

'Send the web service request ...

```
BodyTxt = getResponse(strURL)
```

'Parse the returned text for strSearch value ...

```
If InStr(1, BodyTxt, strSearch, vbTextCompare) = 0 Then
```

```
    Temp = -1# 'strSearch is absent in the response ... Return -1 ...
```

```
Else
```

```
    Temp = Mid(BodyTxt, InStr(1, BodyTxt, strSearch) + Len(strSearch))
```

```
    Temp = Mid(Temp, 1, InStr(1, Temp, "<") - 1)
```

'See if units is meters or kilometers ...

```
If InStr(Temp, "km") <> 0 Then
```

```
    Temp = Val(Temp) * 0.00062137 * 1000 'Convert km to mile ...
```

```
Else
```

```
    Temp = Val(Temp) * 0.00062137 'Convert m to mile ...
```

```
End If
```

```
End If
```

'Return Temp ...

```
fnGetRoutingDistanceHERE_API = Temp
```

```
End Function
```

