

Project 4: ADTs and OLAP

Course 1-02-326: Database Systems Engineering

Due June 27, 2010

In this assignment you will use two tools which we have studied in class - User Defined Types in MS SQL Server 2005 and the data analysis functions for OLAP.

1 Overview: Schema and Custom Types

The purpose of this assignment is to design and implement a tool which will store long term temperature data for a number of cities. The temperature information will be stored in a data warehouse with the following schema:

```
CREATE TABLE Times (timeid INTEGER PRIMARY KEY, date VARCHAR(20),  
week INTEGER, month INTEGER, quarter INTEGER, year INTEGER);
```

```
CREATE TABLE Locations (locid INTEGER PRIMARY KEY,  
country VARCHAR(20), region VARCHAR(20), locality VARCHAR(20),  
latitude Coordinate, longitude Coordinate);
```

```
CREATE TABLE Temperatures (locid INTEGER, timeid INTEGER,  
temperature Temperature, PRIMARY KEY (locid, timeid),  
FOREIGN KEY (locid) REFERENCES Locations,  
FOREIGN KEY (timeid) REFERENCES Times);
```

Note that the schema is in the standard “star schema” for data warehouses. The dimensions are time and location. Each dimension row has a number of levels to enable easy analysis and comparison at different levels. Notice that there are two new types which have been introduced above: **Temperature and Coordinate**.

1.1 Temperature Type

Temperature is a custom type which represents a temperature value. It can be displayed in Kelvin, Celsius, and Fahrenheit. The temperature can be displayed in the different format using the custom properties Kelvin, Celsius, and Fahrenheit. The formulas for conversion between the formats are well known:

- Celsius = Kelvin -273
- Fahrenheit = $(\frac{9}{5} \times \text{Celsius}) + 32$

Temperature should be able to be built using a Parse() on a string value which ends in either “K”, “C”, or “F”. If it ends in “K”, the string (excluding the final character) is interpreted to be a Kelvin temperature. If it ends in “C”, the string (excluding the final character) is interpreted to be a Celsius temperature. If it ends in “F”, the string (excluding the final character) is interpreted to be a Fahrenheit temperature.

1.2 Coordinate Type

Coordinate is a custom type which represents a coordinate value (latitude or longitude). Each location on the Earth can be described using a (latitude, longitude) pair. There are two common coordinate formats - decimal and degrees, minutes, seconds (DMS).

- In decimal format, each coordinate is represented by a single real number. The whole part of the decimal is the number of degrees (fractions of 360 from the total distance). The fractional part of the decimal represents fractions of degrees. For example, 41.836944 represents a position precisely 41.836944 degrees from the origin.
- In DMS format, the coordinate is divided into three parts - (whole) degrees ($^{\circ}$), (whole) minutes ($'$), (real) seconds ($''$). Degrees are fractions of 360 from the total distance as in decimal format. Minutes are fractions of 60 of a single degree. Seconds are fractions of 3600 of a single degree. For example, $41^{\circ}50'13''$ represents a position which is $41 + \frac{50}{60} + \frac{13}{3600}$ degrees from the origin.

To convert between the formats, use the following rules:

- DMS to Decimal: Degrees + $\frac{\text{minutes}}{60}$ + $\frac{\text{seconds}}{3600}$.
- Decimal to DMS: For example, 41.836944.
 1. The Degrees value is the whole number value of the Decimal format (41)
 2. Multiply the fractional part of the Decimal format (0.836944) by 60 (=50.21664). The whole number value is the Minutes (50).
 3. Multiply the fractional part of the previous value (0.21664) by 60 (=12.9984). That is the seconds value (12.9984)

It is common to represent locations on the Earth in terms of distance from the **origin** where the Equator meets the Prime Meridian. This location is denoted (0, 0). All locations North of the origin are given positive (> 0) latitude values. All those South of the origin are given negative (< 0) latitude values. All locations East of the origin are given positive longitude values. All locations West of the origin are given negative longitude values. See Figure 1 for a graphical representation.

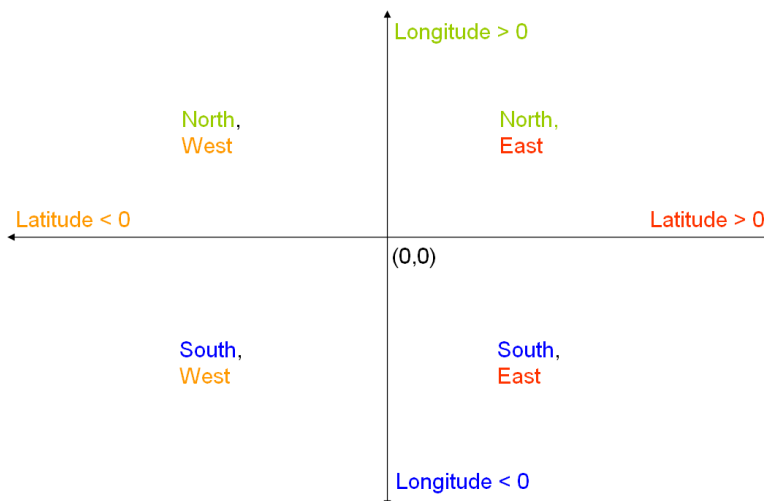


Figure 1: Origin and North/South/East/West Latitude and Longitude Values

Coordinate values must be able to be shown in both the DMS and decimal formats using the custom properties `DMS` and `DecimalDegrees`. There are also custom accessor functions to get parts of the coordinate in DMS format - `GetDegrees()`, `GetDegreesMinutes()`, `GetDegreesMinutesSeconds()`.

Coordinate should be able to be built using `Parse()` on a decimal number (in which case it is interpreted as a decimal degree value) or on a string (in which case it is interpreted as a DMS format coordinate).

1.3 Table Data to Insert

To help perform analysis, I have prepared tools and data sets for you to test your tables and types.

The Locations data is provided in an Excel file. It contains location information for 40 major cities in the USA and Canada along with their (accurate) latitude and longitude values in Decimal format.

I have provided simple command line tools for creating as many Times and Temperatures insert lines as you wish. The source code and executables are provided. Feel free to modify the source code and recompile it as you wish.

I have placed the Excel file, the source code, and the executables mentioned above in a single ZIP file called `326-Assignment4-DataTools.zip` on the course web page and Telem.

2 What to do

Your task in this assignment is to develop an OLAP reporting tool which uses the User Defined Types mentioned above.

2.1 Types to Define

You must define and write the code for two User Defined Types:

1. Temperature with the following functions and properties:
 - `.ToString()` - shows the temperature in a native format (Kelvin)
 - `.Kelvin` - shows the temperatures in Kelvin
 - `.Celsius` - shows the temperature in Celsius
 - `.Fahrenheit` - shows the temperature in Fahrenheit
 - `.Parse(SqlString s)` - creates a Temperature from a numeric string which ends in 'C', 'K', or 'F'
 - `.Addition(int v)` - adds v degrees Kelvin to the temperature
 - `.Addition(string s)` - converts s to a Temperature and adds it to the temperature
 - `.Subtraction(int v)` - subtracts v degrees Kelvin from the temperature (you don't need to check that the result remains > 0)
 - `.Subtraction(string s)` - converts s to a Temperature and subtracts it from the temperature (you don't need to check that the result remains > 0)
2. Coordinate with the following functions and properties:
 - `.DecimalDegrees` - shows the coordinate in Decimal format
 - `.DMS` - shows the coordinate in Degrees, Minutes, Seconds format
 - `.ToString()` - prints the coordinate in a string in Decimal format (**Note:** This is identical to the `.DecimalDegrees` property above, but is included for completeness)
 - `.Parse(SqlString s)` - creates a Coordinate from a string which may be in Decimal format (a single real number) or Degrees, Minutes Seconds format (*ex.* `123d45m67.9s`).
 - `.GetDegrees()` - returns just the (whole) degree portion of the coordinate

- `.GetDegreesMinutes()` - returns the (whole) degrees and (whole) minutes portion of the coordinate in Degrees, Minutes, Seconds format
- `.GetDegreesMinutesSeconds()` - returns the (whole) degrees, (whole) minutes, and (real) seconds of the coordinate in Degrees, Minutes, Seconds format (**Note:** This is identical to the `.DMS` property above, but it is included for completeness).
- `.Addition(double d)` - adds d (real) degrees to the coordinate in Decimal format
- `.Addition(int deg, int min, double sec)` - adds deg degrees, min minutes, and sec seconds to the coordinate in Degrees, Minutes, Seconds format
- `.Subtraction(double d)` - subtracts d (real) degrees from the coordinate in Decimal format
- `.Subtraction(int deg, int min, double sec)` - subtracts deg degrees, min minutes, and sec seconds from the coordinate in Degrees, Minutes, Seconds format

Your types must be correctly defined with native storage (`isByteOrdered=true`) and sorting (`.ToString()`). If they are correctly defined you should be able to use the table definitions above and the automated insertion tools that I have prepared.

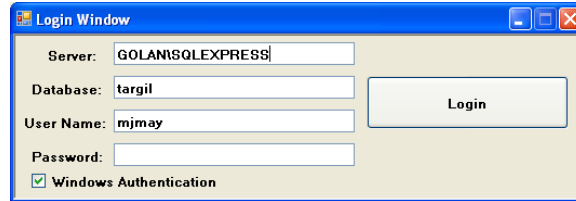
2.2 Queries to Automate

Your tool will enable the execution of a series of summary queries over the temperature data which you have entered in. The queries you write will use some of the tools and commands we have studied in class. Some of them are simply aggregation queries, but are representative of queries commonly performed in OLAP databases.

1. Show the top/bottom n temperatures for all countries/regions/localities over the last m years/quarters/months/weeks in Fahrenheit/Celsius/Kelvin
 - Example: Show the **top 10** temperatures for all **localities** over the last **5 months** in **Fahrenheit**
2. For each year/quarter/month/week, show the country/region/locality with the highest/lowest absolute temperature in Fahrenheit/Celsius/Kelvin
 - Example: For each **month**, show the **region** with the **highest** absolute temperature in **Celsius**
3. For a given locality, show the highest/lowest temperature among all localities in the same Degree/DegreeMinute Latitude/Longitude/LatitudeLongitude for a given Year/YearQuarter/YearQuarterMonth/YearQuarterMonthWeek in Fahrenheit/Celsius/Kelvin
 - Example: For **'Houston'**, show the **highest** temperature among all localities in the same **Degree Latitude** for Year 2010 in Celsius
4. Show all temperatures for a given country/region/locality between the Date d1 and Date d2 in Fahrenheit/Celsius/Kelvin
 - Example: Show all the temperatures for the region **'TX'** between the dates **'1/1/2010'** and **'1/9/2010'** in **Fahrenheit**
5. For each country/region/locality show the percentage change in average temperature over all years/quarters/months/weeks/daays in Fahrenheit/Celsius/Kelvin
 - Example: For each **locality**, show the percentage change in average temperature over all **quarters** in **Kelvin**

2.3 Display Tool

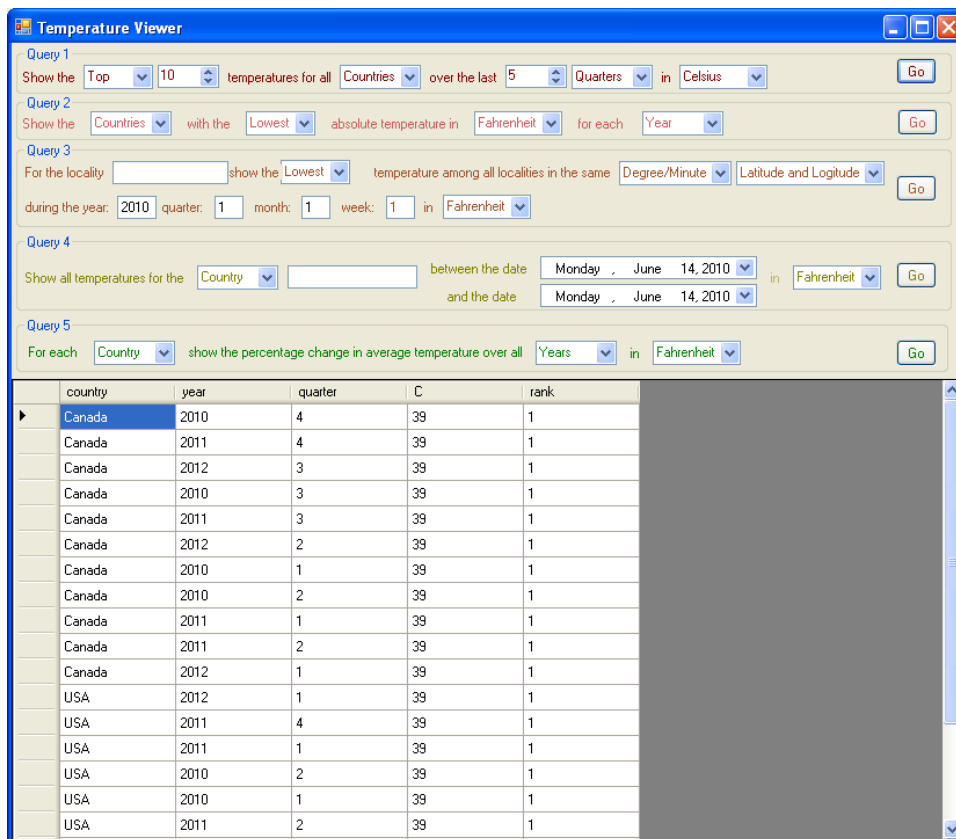
You must develop a tool which will allow a user to automatically generate the above reports by setting the appropriate parameters. The tool will be accessed using an MS SQL connection string as you have developed for the previous assignment (and saw in targil). A sample login screen is shown in Figure 2.



A screenshot of a 'Login Window' application. It features a blue title bar and a light beige background. The window contains four text input fields: 'Server' (with 'GOLANISQLEXPRESS' entered), 'Database' (with 'targil' entered), 'User Name' (with 'mjmay' entered), and 'Password' (empty). A 'Login' button is positioned to the right of the 'User Name' field. At the bottom left, there is a checked checkbox labeled 'Windows Authentication'.

Figure 2: Sample Login Window

Once logged in, the tool must enable queries on an MS SQL Server instance which contains the tables as described above. A sample screen shot of the tool I have developed for this is shown in Figure 3. The buttons labeled “Go” execute the relevant queries and display the results in the `DataGridView` below. The designer code for the GUI shown can be found on the course web site. You are free to use a different design so long as it enables the correct execution of the above queries with the parameter flexibility required.



A screenshot of the 'Temperature Viewer' application. It has a blue title bar and a light beige background. The interface is divided into five query sections, each with a 'Go' button. Query 1: 'Show the Top 10 temperatures for all Countries over the last 5 Quarters in Celsius'. Query 2: 'Show the Countries with the Lowest absolute temperature in Fahrenheit for each Year'. Query 3: 'For the locality [empty] show the Lowest temperature among all localities in the same Degree/Minute Latitude and Longitude during the year: 2010 quarter: 1 month: 1 week: 1 in Fahrenheit'. Query 4: 'Show all temperatures for the Country [empty] between the date Monday, June 14, 2010 and the date Monday, June 14, 2010 in Fahrenheit'. Query 5: 'For each Country show the percentage change in average temperature over all Years in Fahrenheit'. Below the queries is a `DataGridView` with the following data:

	country	year	quarter	C	rank
▶	Canada	2010	4	39	1
	Canada	2011	4	39	1
	Canada	2012	3	39	1
	Canada	2010	3	39	1
	Canada	2011	3	39	1
	Canada	2012	2	39	1
	Canada	2010	1	39	1
	Canada	2010	2	39	1
	Canada	2011	1	39	1
	Canada	2011	2	39	1
	Canada	2012	1	39	1
	USA	2012	1	39	1
	USA	2011	4	39	1
	USA	2011	1	39	1
	USA	2010	2	39	1
	USA	2010	1	39	1
	USA	2011	2	39	1

Figure 3: Sample Screen Shot for ADT OLAP Tool

3 Grading and Submission

The assignment is worth 100 points total. Points will be assigned as follows:

1. Correct login and connection strings. 5 points
2. Correct design and definition of the User Defined Types (Temperature and Coordinate). 30 points
3. Correct execution of queries - 60 points (12 points each)
4. General robustness. 5 points.

You may work in groups of **up to 3 students**. I strongly advise students to not work alone. There is a significant amount of coding work here, more than I estimate a single student could comfortably do alone.

You must submit your work via email or Telem by June 27, 2010 at 11:59pm. Keep in mind the submission rules and late penalties listed in the syllabus.

Submit your work as a single Zip or RAR file with the following contents:

- The Visual Studio C# or Visual Basic project file
- All related source code files necessary to compile the project
- A compiled executable file
- A README file which contains the names of the students who worked on the project, the number of hours spent working on it, and any notes or documentation necessary to understand and use the application.

I recommend using email ([ise326@gmail](mailto:ise326@gmail.com)) since Telem will not let you easily upload ZIP files.