1 Geodistance Function

In this task the goal is to implement a user defined function (UDF) to compute the distance (in kilometers) between two GPS coordinates. In the Mondial database GPS coordinates are stored in the form of the custom data type `GeoCoord` that consists of two attributes `Latitude` and `Longitude`.

Let \((\phi_1, \lambda_1)\) and \((\phi_2, \lambda_2)\) be two GPS coordinates, where \(\phi_1, \phi_2\) and \(\lambda_1, \lambda_2\) are latitudes and longitudes in degrees, respectively. The geodistance \(d\) between these two points is computed as follows:

\[
R = 6371 \quad \text{(radius of the earth in kilometers)}
\]

\[
c = \frac{2\pi}{360} \quad \text{(factor to convert degrees to radians)}
\]

\[
a = \sin^2\left(\frac{c(\phi_2 - \phi_1)}{2}\right) + \cos(c \cdot \phi_1) \times \cos(c \cdot \phi_2) \times \sin^2\left(\frac{c(\lambda_2 - \lambda_1)}{2}\right)
\]

\[
d = R \times 2 \times \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)
\]

The four-quadrant inverse tangent function `atan2(x, y)` is a variant of the arctangent trigonometric function, where the signs of \(x\) and \(y\) are used to determine the quadrant of the resulting angle.

Tasks:

1. Develop and write down a UDF that implements the above formula and returns distance \(d\). The UDF’s signature is

   `CREATE OR REPLACE FUNCTION geoDist(c1 GeoCoord, c2 GeoCoord)
   RETURNS NUMERIC
   AS $$
   DECLARE
   Radius NUMERIC;
   a NUMERIC;
   c NUMERIC;
   BEGIN
   Radius := 6371.0;  -- radius of the earth in meters
   c := (2*pi()) / 360;  -- to convert degrees to radians
   a := power(sin(c*(c2.Latitude - c1.Latitude) / 2), 2) + (cos(c * c1.Latitude)
   * cos(c * c2.Latitude)
   * power(sin(c*(c2.Longitude - c1.Longitude) / 2), 2));
   RETURN Radius * 2 * atan2(sqrt(a), sqrt(1-a));
   END; $$ LANGUAGE plpgsql;
   $$
   ```

See PostgreSQL’s documentation[1] for a description of the required math functions.

Solution:

`CREATE OR REPLACE FUNCTION geoDist(c1 GeoCoord, c2 GeoCoord)
RETURNS NUMERIC AS $$
DECLARE
  Radius NUMERIC;
  a NUMERIC;
  c NUMERIC;
BEGIN
  Radius := 6371.0;  -- radius of the earth in meters
  c := (2*pi()) / 360;  -- to convert degrees to radians
  a := power(sin(c*(c2.Latitude - c1.Latitude) / 2), 2) + (cos(c * c1.Latitude)
  * cos(c * c2.Latitude)
  * power(sin(c*(c2.Longitude - c1.Longitude) / 2), 2));
  RETURN Radius * 2 * atan2(sqrt(a), sqrt(1-a));
END; $$ LANGUAGE plpgsql;`
2. You can check your function with the following example query and its expected resulting table.

```
SELECT geoDist((
    SELECT Coordinates FROM island WHERE Name = 'Kos'),
    (SELECT Coordinates FROM island WHERE Name = 'Samos'))
AS Dist;
```

<table>
<thead>
<tr>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>103.538090149668</td>
</tr>
</tbody>
</table>

3. Execute the following query and write down its result

```
SELECT geoDist((
    SELECT Coordinates FROM island WHERE Name = 'Guam'),
    (SELECT Coordinates FROM island WHERE Name = 'Korfu'))
AS Dist;
```

**Solution:**

<table>
<thead>
<tr>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>11823.0473909191</td>
</tr>
</tbody>
</table>

## 2 Computing Nearest Islands

Computing the nearest neighbors of a spatial object (e.g. an island) is a common task. The following UDF receives the name `IName` of an island as single input parameter and returns a table of three attributes. The function returns `IName`, the names of the 3 nearest islands according to the previously defined distance function, and their distance to this island `IName`.

**Task:** Execute the following function definition, which will be needed in the tasks to follow.

```
CREATE OR REPLACE FUNCTION nearestIslands(IName VARCHAR(35))
RETURNS TABLE(Name VARCHAR(35), Neighbor VARCHAR(35), Dist NUMERIC)
AS $$
BEGIN
    RETURN QUERY
    SELECT i1.Name AS Name, i2.Name AS Neighbor,
           geoDist(i1.Coordinates, i2.Coordinates) AS Dist
    FROM island i1, island i2
    WHERE i1.Name = IName AND i2.Name <> i1.Name
    ORDER BY geoDist(i1.Coordinates, i2.Coordinates), i1.Name
    LIMIT 3;
END; $$ LANGUAGE plpgsql;
```

## 3 Caching Nearest Islands

Computing nearest neighbors is a common, but expensive operation. The goal of this task is to cache the results of the UDF created in the previous task to speed up repeated invocations of the same nearest neighbor query.

**Tasks:**
1. Write down a `CREATE TABLE` statement that creates a new table called `cachedNearestIslands`, which has the same schema as the table that function `nearestIslands` returns.

   Solution:
   ```sql
   CREATE TABLE cachedNearestIslands(
       Name VARCHAR(35),
       Neighbor VARCHAR(35),
       Dist NUMERIC
   );
   ```

2. Develop and write down a UDF that takes one parameter and has the following signature:

   ```sql
   CREATE OR REPLACE FUNCTION fastNearestIslands(Iname VARCHAR(35))
   RETURNS TABLE(Name VARCHAR(35), Neighbor VARCHAR(35), Dist NUMERIC)
   AS $$
   BEGIN
   RETURN QUERY
       SELECT * FROM cachedNearestIslands c
       WHERE c.Name = Iname;

   IF FOUND THEN
       RAISE NOTICE 'Cache hit for island "%"', Iname;
   ELSE
       RAISE NOTICE 'Cache miss for island "%"', Iname;

       INSERT INTO cachedNearestIslands
       SELECT * FROM nearestIslands(Iname);

       RETURN QUERY
       SELECT * FROM cachedNearestIslands c
       WHERE c.Name = Iname;
   END IF;
   END $$ LANGUAGE plpgsql;
   ```

   Note: The PostgreSQL documentation for PL/pgSQL errors and messages is available [here](https://www.postgresql.org/docs/9.4/static/plpgsql-errors-and-messages.html).
3. Execute the following query and write down any printed log message and the resulting table

```sql
SELECT * FROM fastNearestIslands('Guam');
```

**Solution:**

```
NOTICE: Cache miss for island "Guam"
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Neighbor</th>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Guam'</td>
<td>'Rota'</td>
<td>201.912295167509</td>
</tr>
<tr>
<td>'Guam'</td>
<td>'Tinian'</td>
<td>202.643342613215</td>
</tr>
<tr>
<td>'Guam'</td>
<td>'Saipan'</td>
<td>229.911062890579</td>
</tr>
</tbody>
</table>

4 Cache Refresh

When table `island` is modified, the actual nearest neighbors might change and differ from the cached ones. The goal of this task is to develop a trigger to automatically refresh the cache upon insertions on table `island`.

**Tasks:**

1. Write a trigger that refreshes the `cachedNearestIslands` table on every `insert` of a new tuple into the `island` relation. Refreshing means that for every cached island (attribute Name in `cachedNearestIslands`) (a) a message should be logged on the NOTICE level saying ‘Refreshing nearest neighbors for island “Name”’ (replacing Name with the actual name of the island), and (b) its nearest islands are recomputed.

**Solution:**

```sql
CREATE OR REPLACE FUNCTION cacheRefresh() RETURNS TRIGGER AS $$
DECLARE
  tuple Record;
BEGIN
  FOR tuple IN
    SELECT DISTINCT Name FROM cachedNearestIslands
  LOOP
    RAISE NOTICE 'Refreshing nearest neighbors for island "%"', tuple.Name;
    DELETE FROM cachedNearestIslands WHERE Name = tuple.Name;
    INSERT INTO cachedNearestIslands
    SELECT * FROM nearestIslands(tuple.Name);
  END LOOP;
  RETURN NEW;
END$$ LANGUAGE plpgsql;

CREATE TRIGGER cacheRefreshTrigger
AFTER INSERT ON island
FOR EACH STATEMENT EXECUTE PROCEDURE cacheRefresh();
```
Notice that the `SELECT` statement in line 7 is first fully executed before the body of the loop is executed. This is due to SQL’s logical update semantics.

Further notice that by using `DISTINCT` in line 7 we avoid that the nearest neighbors for a particular island are recomputed more than one.

2. Execute the following two queries and write down any printed log message and the resulting table.

```sql
INSERT INTO island VALUES ('Test1', 'Test1', 100, 100, NULL, (144.9,13.1));

SELECT * FROM fastNearestIslands('Guam');
```

**Solution:**

```
NOTICE: Refreshing nearest neighbors for island "Guam"
NOTICE: Cache hit for island "Guam"
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Neighbor</th>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guam</td>
<td>Test1</td>
<td>39.7665432593773</td>
</tr>
<tr>
<td>Guam</td>
<td>Rota</td>
<td>201.912295167509</td>
</tr>
<tr>
<td>Guam</td>
<td>Tinian</td>
<td>202.643342613215</td>
</tr>
</tbody>
</table>