doi:10.2498/cit.1002345

A Proposal to Expand the Community of Users Able to Process Historical Rainfall Data by Means of the Today Available Open Source Libraries

Paolino Di Felice[1], Luca Finocchio[2], Daniele Leombruni[2] and Vittoriano Muttillo[2]

[1] Department of Ingegneria Industriale e dell'Informazione ed Economia, University of L'Aquila, Italy
[2] Master students in Engineering Informatics, University of L'Aquila, Italy

The paper presents a software architecture based on open source technologies, implemented by the authors in an experience of processing spatio-temporal data gathered by rain gauges spread across two regions of central Italy. The interest in the automatic processing of data about precipitation is widespread, however, today only an inner circle of stakeholders can think of taking advantage of the available open source libraries because a strong programming skill is required to use them. It is opinion of the authors that the implemented software architecture is suitable for expanding the community of users able to process "by themselves" historical precipitation data. The centre of the architecture is the technology of the spatial database management systems. They offer full support for the creation and management of a spatial database suitable to store the rainfall data usually spread out in several files. Moreover, they allow adding to such a repository a large set of ad hoc "*objects*" oriented to carrying out spatio-temporal computations on the precipitation. The stakeholders are only required to familiarize with the database's objects and invoke their execution. A large part of the paper is devoted to show *how* the adopted conceptual setting can assist nontechnical users in carrying out personalized computations on rain gauges data. The computing needs posed by the experience described in this paper are common to many other areas of high social impact that involve spatio-temporal data, hence we believe that the implemented framework can be exported to them, keeping unaltered operational effectiveness.

Keywords: open source software, DBMS, SQL, PostGIS, R language, software architecture, user defined function, precipitation retrieval

1. Introduction

The wise management of water reserves, the prevention of natural disasters related to excessive rains, and the choise of the best land use in agriculture are few issues of primary importance to most countries. The modeling of these scenarios and the efficient management of the huge amount of spatio-temporal data involved are very complex problems that must be addressed with methods and tools from the areas of computing and information technology. Some of the recent contributions in this direction are: (Lu and Wong, 2008; Krishnakumar et al., 2009; Li et al., 2011; Rowhania et al., 2011; Tabari and Talaee, 2011; Ahani et al., 2012; Bostan et al., 2012; Keblouti et al., 2012; Chen and Liu, 2012).

Unfortunately, in most of the IT approaches appeared so far the input data are dispersed in a multitude of independent files, sometimes conflicting even in their format. In this regard (Carleton et al, 2005) state the following. "Typically, hydrologic data are stored in a spreadsheet. To keep data organized in a spreadsheet, separate files are often created by the user to hold data from different locations and years. This results in an ever increasing number of files, each of which have to be individually managed whenever new data are added or new calculations desired. Problems also arise when attempting to compare data between two or more

individual spreadsheet files. Spreadsheets also make it difficult to analyze data categorically, such as on a daily or monthly time step."

To overcome this limit, the software architecture we present in this paper, and implemented in a recent experience of storage, cleaning, processing and visualization of spatio-temporal data collected by rain gauges located on two regions of central Italy, puts at the centre the technology of the Spatial Database Management Systems (SDBMSs). This technology has dramatically increased the effectivness and efficiency in carrying out many everyday activties involving large structured datasets.

Our software architecture is based exclusively on open source technologies. Since several years ago there has been a consolidation of open source software able to manipulate spatio-temporal data. This fact, together with the appearance of international no-profit foundations (e.g., Open Source Geospatial Foundation – OSGeo), indicates that the interest in open source software is expected to grow significantly in the near future. Steiniger and Bocher (2009) recognize four indicators of this trend: (1) increasing number of projects run using open source GISs, (2) increasing financial support by government agencies, (3) increasing download rates, and (4) increasing number of use-cases. With regard to the geostatistical computations of rainfall data, in particular, the use of the R environment (Pebesma, 2004), extended with libraries such as *gstat* and *raster*, today appears as the best choice. Unfortunately, as stressed by Hengl (2009) – page 90, "the main problem with R is that each step must be run via a command line, which means that the analyst must really be an R expert". This circumstance limits the use of this valuable software tool to subjects with strong programming aptitude.

Thanks to the possibility of formulating "powerful" spatial queries at a "high level" of abstraction, SDBMSs have the further merit of greatly extending the community of those who can interact with such a software technology to perform relevant spatial analysis without owning a specific IT skill. In the experience of computing of spatio-temporal data reported in this paper, the query *expressiveness* is obtained by placing side by side to the basic SQL operators and to the built-in functions of the SDBMS, "database objects" created ad hoc (technically they are User Defined Functions – UDFs), while the *abstraction* is obtained by masking to the end user the logical structure of the underlying database (for example, through the mechanism of the *views*), as well as the functions called in the query. From the stakeholders involved is only required to familiarize with the objects that the database exposes and invoke their execution. This level of competence can be achieved after a short on-the-job training of the nontechnical personal.

The paper is divided in two parts. The first part: a) proposes the technological solution setting and lists its several advantages over the current ways of processing historical data about precipitation (Section 2), b) describes the reference format of the rainfall data, the design of a Spatial DataBase (SDB) adequate to store the data about the geographical area of interest, the rain gauges spread over it, and the collected measurements (Section 3), c) gives details about the structuring of a number of packages that implement the computing requirements posed by the application domain we refer to (Section 4), and, finally, d) discusses the related work (Section 5). The second part of the paper (Section 6) shows, through a case study about rainfall data recorded by rain gauges located in central Italy, the simplicity with which, working from the inside the console of the SDBMS, one could take advantage of the many features offered by the packages we have developed (and displayed as database objects) to implement the phases ranging from the storage of the measurements in the SDB, to their cleaning and processing, till the production of maps.

2. The Solution Setting

Figure 1 shows the reference software architecture of our solution setting. It can be interpreted at two different levels: one *abstract*, the other *concrete*. The abstract level highlights the basic elements of the solution: a SDBMS and a programming language oriented to geostatistics. Both elements are on top of a SDB containing the data about the geographic area of interest, the rain gauges located in it and the measurements collected. The concrete level, vice versa, details the software technologies chosen, i.e., PostgreSQL/PostGIS and PL/R (with the libraries *gstat* and *raster*).

The central element of the architecture of Figure 1 is the technology of the SDBMSs. It offers enormous advantages over the current ways of processing the historical data about precipitation; in detail, it allows us:

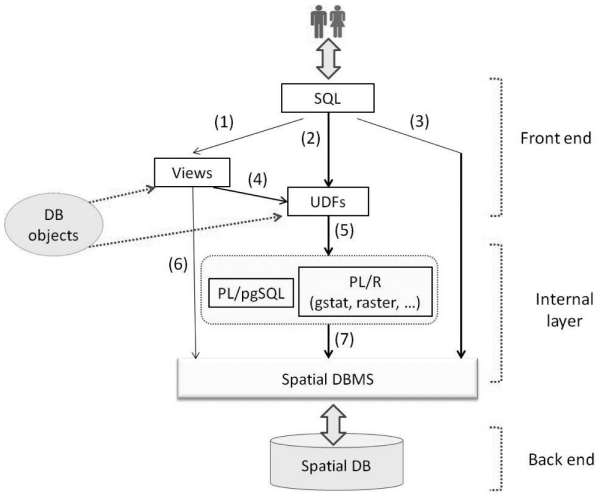

Figure 1. The reference software architecture.

1. to keep together, in a single container, all the historical measurements. Data typically dispersed in a multitude of independent files, sometimes conflicting even in their format;
2. to limit the access to the data only to people holding explicit granting privileges;
3. to call the spatial operators of the SQL language as well as the spatial built-in UDFs, when querying the SDB. UDFs are a relevant capability of SDBMSs. For example, PostgreSQL/PostGIS (ver. 9.2, PostGIS model: postgis_21_sample) offers 1092 built-in UDFs;
4. to code ad hoc views and UDFs (to be added to the built-in UDFs) to be exposed as database objects, both joinable in SQL queries of extraordinary expressiveness;
5. to transform the end users' rainfall data processing needs from programming activities, as it happens today, to querying activities against the SDB, with considerable masking of technicalities and, therefore, potentially widening the number of those who in the future could face such difficulties with profit. In fact, by taking advantage of the geostatistical and graphical functionalities of the R programming environment (and of the libraries linked to it, over all *gstat* and *raster*), with which the SDBMS communicates effectively, it is easy to greatly extend the capabilities of the SQL-pure UDFs (i.e., functions written by embedding SQL commands into the internal language of the SDBMS), greatly simplifying the job of the end users.

Figure 1 shows the paths through which users can interact with the SDB. In each path, it is required to write SQL queries to be executed in the console of the SDBMS. For users without any technical skill, the interaction with the rainfall data takes place by invoking the database objects (either path 1 or 2), while path (3) is for skilled users. Path (1) may involve objects of type *view* (path 1-6), or views and UDFs together (path 1-4-5-7), while path (2-5-7) involves only UDFs. If the user invokes in the query some UDFs, he/she, however, does not perceive their internal encoding, and even the coding language (either PL/pgSQL or PL/R).

To learn about the merits of the R environment, refer to (Hengl, 2009). The literature already proposes examples of usage of R to develop packages (e.g., Cannon, 2011).

The communication between PostgreSQL and R may take place either by keeping them separate, or by using R from inside PostgreSQL. Choosing the first option, they communicate by importing/exporting shapefiles: so, one can, for example, run a query in PostgreSQL, export the result as a shape, load it into R and, eventually run (in this environment) the processing he is interested in. But because, as already mentioned, work with R implies that the analyst must really be an R expert, we decided to adopt the second option in which the use of R is hidden to the end users of the SDB, therefore, they may not be programmers.

PostgreSQL allows us to interact with the R environment by using PL/R, i.e., a procedural language suitable for writing PostgreSQL functions in the programming language R. PL/R provides most of the R's functionalities.

3. Format of the Precipitation Data and Structure of the Spatial Database

In Italy, data about the precipitation over the territory are collected by different regional and national agencies without, however, their sharing either a standard methodology or a common

data format. Burnette and Stahle (2013) report a similar situation for the United States. In this paper, we will refer to the data format adopted by the Italian Department of Civil Protection, which is a structure of the Presidency of the Council of the Ministers (`http://www.protezionecivile.gov.it`). Section 6.1 explains how our packages intercept the format heterogeneity in case the precipitation data come from a different source, allowing, therefore, to keep the structure of the SDB unaltered.

3.1. Format of the Precipitation Data

The rainfall data are collected in two categories of files. The first one consists of a single file about the stations of measures distributed over the land, while the second category is composed of many files as the months to which the measurements refer to. The rain gauges measure the precipitation volume (in mm) each 15 minutes using a counter that is incremented by one each time it detects 0.1 mm of rain. The rain gauge calculates the precipitation accumulated in the last quarter of an hour by subtracting from the current value of the counter the value of the counter at the previous measurement, and dividing the result by 10 (to express the value in mm).

The metadata about the rain gauges are described by the tuple: <*id*, *name*, *station number*, *municipality*, *region*, *geographical position*> the latter expressed as latitude and longitude. The metadata about the rainfall measurements are described by the tuple: <*rain gauge id*, *data*, *hour*, *rainfall in 15 minutes* (mm), *counter*>.

3.2. Structure of the SDB

The SDB (called `Rainfall_measurement`) suitable to hold the data of interest is composed of the four tables of Figure 2. `region` and `district` refer, respectively, to the region and the municipality where the meteorological stations are located.

The SQL/DDL statements about the four tables of Figure 2 are given in the Appendix.

4. Organization of the Packages

The precipitation context raises many processing requirements. We discuss them in the following. The initial need consists of loading the measurements recorded by the rain gauges into the SDB. Subsequently, it is useful to perform an analysis about the distribution of the stations over the land. In fact, the sampling density plays a significant role in the performance of the spatial interpolation methods. Li and Heap (2008) report that "when data are sparse, the underlying assumptions about the variation among samples may differ and the choice of a spatial interpolation method and parameters may become critical."

The validation of the data about the precipitation is another crucial issue for increasing their quality. Data quality is among the major factors that affect the performance of the spatial interpolation methods (Li and Heap, 2008). Data noise can negatively affect the output of those methods.

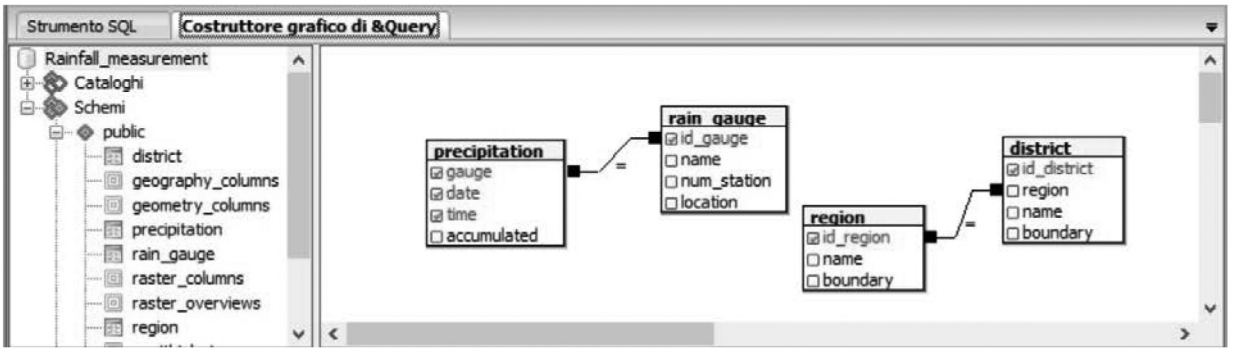

Figure 2. The four tables of the SDB. The red attributes are primary keys. The arcs connect the primary key to the corresponding foreign key.

The construction of diagrams about the trend of the rainfall for time intervals selected on an hourly, daily, weekly, monthly or yearly basis is another crucial requirement.

Interpolation of the measured data over the geographical area with the ultimate goal of getting maps about the rainfall is fundamental. Interpolation techniques widely used are the inverse distance interpolation (IDW) method and the Kriging (Li and Heap, 2008). The cross validation of the interpolation carried out is mandatory to evaluate the results.

Table 1 brings together the processing requirements and the name of the corresponding package of PostgreSQL UDFs that implements them.

The *Auxiliary* package contains 34 functions created to support those contained in the other five packages. Table 2 lists the 34 functions divided according to the used coding language (namely: PL/R or PL/pgSQL). The names of these functions are left intentionally generic (UDF_0, UDF_1,...) so as not to create confusion or conflict with the name of the functions that invoke them.

Table 3 shows, for each package, the number of internal UDFs, the total number of lines of code, the number of UDFs of the package *Auxiliary* called. From Table 3 we learn about the massive use that the UDFs of the various packages make of the UDFs of the *Auxiliary* package.

The PL/pgSQL code refers to version 9.2 of PostgreSQL and to version 1.8 of PostGIS, while the PL/R code refers to version 8.3.0.14.

The processing requirements	**The package of UDFs**
Loading the precipitation data into the SDB Cleaning of the precipitation data Analysis of the distribution of the rain gauges over the *Region*	*Basic*
Construction diagrams about the rainfall for different time intervals (hours, days, weeks, months, years)	*Plotting*
Interpolation of the precipitation data over *Region*	*IDW, Kriging*
Cross validation of the interpolation	*CrossValidation*
	Auxiliary

Table 1. The processing requirements. *Region* denotes a generic geographic area.

Language	**Name of the UDF**
PL/R	UDF_0, UDF_6, ..., UDF_14, *UDF*_16, ..., UDF_23, UDF_27, ..., UDF_31
PL/pgSQL	UDF_1, ..., UDF_5, UDF_15, UDF_24, UDF_25, UDF_26, UDF_32, UDF_33, UDF_34

Table 2. List of the UDFs in the *Auxiliary* package.

Package	**Total # of UDFs**	**Total # of lines of code**	**Total # UDFs of the package Auxiliary called**
Basic	7	205	1
Plotting	20	604	36
IDW	4	206	$(4 * 4 =)16$
Kriging	16	760	$(16 * 4 =)64$
CrossValidation	12	421	$(12 * 2 =)24$
Auxiliary	34	1026	

Table 3. Numbers about the implemented packages.

5. Related Work

(Ward et al., 1996), a pioneering work in the field of geocomputing, outline a computational architecture suitable to deal with the complexity and the diversity of data and processing needs in geoscientific data analysis. One of the key elements of the proposed architecture is the layer about the management of data. The architecture we implemented to carry out the experience about spatio-temporal raing gauges data intercepts this recommendation.

In (Shen et al., 2005), a relational database is introduced as the backbone of a system to deal with hydrological data." Authors attribute three major merits to their system's underlying database-supporting architecture: a) "great flexibility for user manipulation and customization"; b) "the great potentiality deriving from using the built-in database functions"; c) "the database builds a linkage among users' inputs, model data management, and the numerical model by implementing a set of dynamic database queries." The above three merits apply to our solution as well.

In the same year, (Carleton et al., 2005) developed a Microsoft Access database designed to facilitate the storage, retrieval and analysis of hydrologic data as an alternative to the prevalent trend at the time of storing such data in spreadsheets which, as already pointed out, suffer from severe limits. Besides presenting the database structure, (Carleton et al., 2005) provide a set of queries useful to retrieve data that involve calculations, comparisons and basic quality assurance/quality control calculations. Our experience reaffirms the relevance of querying, besides showing a way to deeply enhance its expressiveness by calling ad hoc UDFs and/or views.

Pokorny (2006) remarks that since the 1960s there is an ever increasing environmental demand from customers, authorities and governmental organizations. Then, he states that "A natural idea is that environmental information systems based on advanced database technologies could help to deal with these issues." Our practical experience builds on both the above statements.

6. The Case Study

This section shows the ease of use, as well as the effectiveness, of the UDFs collected in the six packages developed (Table 3), to support the archiving, cleaning and processing of precipitation data up to the production of rainfall maps. The discussion will involve only some of the UDFs.

The case study takes into account the measurements recorded in November 2007 by 137 meteorological stations, equipped with a rain gauge, located in the regions of Lazio and Umbria (central Italy). The dataset about the rain gauges and that about the measurements are in the `.csv` format and follow the pattern described in Section 3.

6.1. SDB Loading

Preliminarily, the `.csv` file about the rain gauges and the file about the measurements were both converted (by means of QGIS) into shape files. The shape files containing the geometry of the Italian regions and municipalities were downloaded from ISTAT (`http://www.istat.it/`). The format of the geometric data in all the shape files was expressed in the WGS 84 Spatial Reference System. Then, we imported (using the appropriate pgAdminIII plug-in) the content of the shape files about the rain gauges, regions and municipalities inside, respectively, tables `rain_gauge`, `region` and `district` of the SDB `Rainfall_measurement`. The *manual* actions mentioned so far are part of what we might call *phase zero*. From here on out, all the actions that will be described are greatly facilitated by the use of the UDFs developed.

To load the contents of the file of the measurements in the table `precipitation` is sufficient to invoke the UDF `import_precipitation()` of the package *Basic* (written in pure SQL – *Appendix*) which, in turn, engages the UDF_0 (written in PL/R) of the *Auxiliary* package. UDF_0 also accounts for replacing with `NULL` missing and negative values. Any changes in the format of the file of the measurements, which in practice is frequent, entail changes to the body of UDF_0, but not to the structure of the SDB which, therefore, remains stable. The loading

is obtained by running (in the query window of pgAdminIII) the SQL command:

```
SELECT import_precipitation('...\
    2007-11csv', '2007-11')
```

This command must be invoked as many times as the months of the measurements to be loaded into the SDB (Section 3).

The invocation of any one of our UDFs has to be made in the same operating mode. In the following we will avoid repeating it.

6.2. Assessment of the Spatial Distribution of the Rainfall Stations Over the Territory

Figure 3 shows that spatial distribution of the 137 meteorological stations is not uniform; in particular, in the case of the Lazio region, there is a high density of rainfall stations around Rome. In Lazio (17,236 km^2) there are 84 stations, in Umbria (8,456 km^2) 47.

Each dot represents one station. Some stations are located slightly outside the two regions (3 in Tuscany, 2 in Abruzzo and 1 in Campania). A confirmation of what appears visually can be obtained by running the query `SELECT count(*) FROM rainGauges_filter('LAZIO')`, where `rainGauges_filter()` is a UDF of the package *Basic*.

As already mentioned (Section 4), the number of the meteorological stations and their distribution over the land impact the performance of the spatial interpolation methods (Li and Heap, 2008). Table 4 shows that each rain gauge in Lazio covers, on the average, about 205 km^2 while in Umbria it covers about 180 km^2. These data confirm that the number of rain gauges in Umbria is more appropriate than that in Lazio. In both cases, the numbers confirm a trend well known in the literature, namely that frequently the number of stations is very low when compared to the size of the territory. For example, Bostan et al. (2012) report on a study about the annual precipitation in Turkey carried out by starting from the measurements acquired in just 225 meteorological stations throughout the entire country (an area of 783.562 km^2). Therefore, it is legitimate to conclude that the situation in Lazio and Umbria is definitely more favorable.

Region	Rain gauges distribution (km^2)
Lazio	204.7
Umbria	179.8

Table 4. Distribution of the rain gauges in Lazio and Umbria. These values are computed by running the (*Basic*) UDF `rainGauges_density()`.

6.3. Data Cleaning

Using the (*Basic*) UDFs `clean_peaks()` and `clean_gauge()` (to be invoked as: `SELECT clean_peaks(3)`; `SELECT clean_gauge(3,30)`) it is

Figure 3. QGIS displaying the contents of the file of the rain gauges.

possible to eliminate the outliers and the suspicious rain gauges as well. Specifically, the first query eliminates from table `precipitation` all the tuples that in the field `accumulated` have a value of the rainfall in 15 minutes greater than 3 mm, while the second query eliminates from table `rain_gauge` the corresponding stations if it happens that they have a percentage of erroneous readings above 30%. The arbitrarily set threshold value of 3 mm in 15 minutes of precipitation can be adjusted to the geographic region of reference.

6.4. Measurements Plotting

Diagrammatic visualization of the measurements collected in the SDB is a relevant need. In this section we show examples of what can be done very easily by invoking the UDFs being part of the package *Plotting*.

The query:

```
SELECT plot_withinHours
('plot_withinHours.pdf', 'Eur',
'2007-11-01', '00:00', '22:15')
```

returns, as a pdf file, the diagram of the rainfall measured by the rain gauge called 'Eur', located in the Trastevere area of Rome, from 00:00 to 22:15 on November 1st, 2007 (Figure 4).

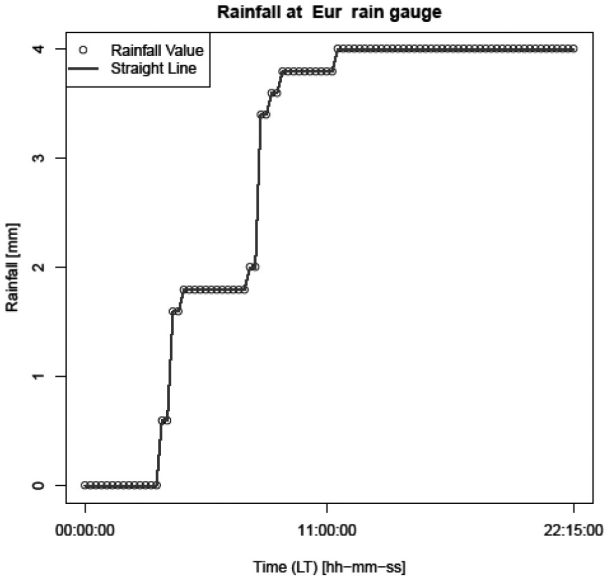

Figure 4. The rainfall measured by the rain gauge Eur on 2007-11-01, between 00:00 and 22:15.

The UDF `plot_withinHours()` (*Appendix*) uses UDF_1 and UDF_7 (*Appendix*). By calling the UDFs: `plot_withinDays()`, `plot_withinMonths()`, and `plot_withinYears()` it is possible to change the granularity of the diagram.

The barplot in Figure 5 shows the rainfall registered at each rain gauge in the Lazio region on 2007-11-01, from 00:00 to 23:00. The diagram is built by running the command:

```
SELECT barPlot_withinHours('2007-11-01',
        '00:00', '23:00',
    'barPlot_withinHours.pdf')
```

Each bar of the graph represents the precipitation accumulated in a rain gauge in the specified twenty-three hours.

To deepen the knowledge about the data behind the diagram of Figure 5, it would be sufficient to make recourse to the querying. Unfortunately, it is not reasonable to assume that nontechnical users have such a skill. But, we can help them by creating the view `rgwithinlazio` that hides in its "body" most details (namely, the need to look at three of the four database tables and join two of them, as well as the need to call the spatial operator `ST_Contains()` and the function `SUM`).

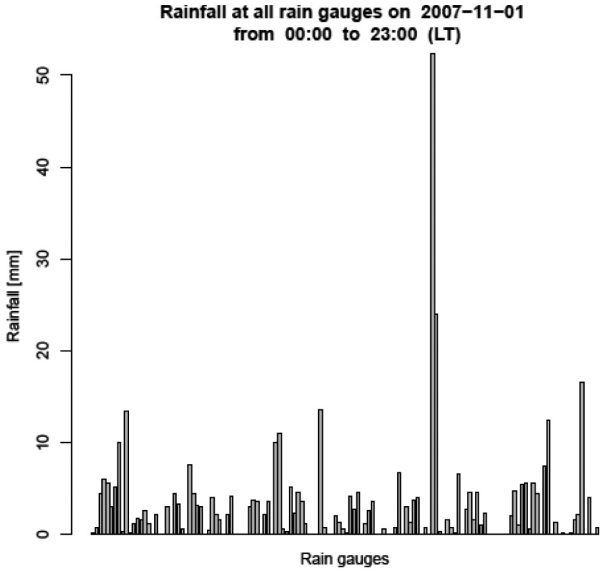

Figure 5. Time series of the precipitation measured by the rain gauges within Lazio on the 1st of November 2007, from 00:00 to 23:00. (A similar histogram may be found in Haberlandt, 2007.)

```
CREATE VIEW rgwithinlazio AS (
    SELECT rg.id_gauge ,rg.name, rg.location, SUM(accumulated) as accumulated
    FROM precipitation AS p, rain_gauge AS rg, region AS r
    WHERE p.gauge=rg.id_gauge AND
        ST_Contains(r.boundary, rg.location)=true AND
        r.name='LAZIO' AND p.date='2007-11-01' AND
        p.time > '00:00' AND p.time < '23:00'
    GROUP BY rg.id_gauge, rg.name, rg.location );
```

After a short on-the-job training, nontechnical users, by calling such a database object, may become able to formulate non-trivial queries such as, for instance, to count the stations that have measured a rainfall above a given threshold (e.g., 13 mm) on 2007-11-01:

```
SELECT count(id_gauge)
FROM rgwithinlazio
WHERE accumulated > 13;
```

or display the identifier, name and geographic position of those stations:

```
SELECT id_gauge name, location
FROM rgwithinlazio
WHERE accumulated > 13;
```

The output of this second query can be displayed in the usual tabular format (Figure 6) as well as a map (Figure 7).

Previous queries | Cancella | Cancella tutto

```
SELECT id_gauge, name, location
FROM   rgwithinlazio
WHERE  accumulated > 13
```

Riquadro di «output»

Output dei dati | Explain | Messaggi | Cronologia

	id_gauge integer	name character varying(50)	location geometry
1	13168	Aurelio	0101000020E610000099
2	15228	Velletri	0101000020E6100000A3
3	18356	Atina	0101000020E6100000CE
4	18365	Colleferro	0101000020E610000079

Figure 6. The table collecting the identifier, name and geographic position of the rain gauges within Lazio that measured, on the 1st of November 2007, a precipitation volume above 13 mm.

Figure 7. The (QGIS) map about the rain gauges within Lazio that measured, on the 1st of November 2007, a precipitation volume above 13 mm.

The UDFs `rainfallPlot_withinHours()`, `rainfallPlot_withinDays()`, `rainfallPlot_withinMonths()`, `rainfallPlot_withinYears()` build diagrams that bring the progressive sum of the precipitation recorded in the time interval specified, as well as that fallen every 15 minutes.

6.5. Interpolation

In the reality, one always has to estimate the value of environmental variables from a limited set of measurements. For this reason many spatial interpolation methods have been developed. IDW and kriging are the best known methods (Li and Heap, 2008). As pointed out by Bostan et al. (2012), spatial interpolation methods have been applied to precipitation measurements on daily, monthly and annual averages. The *IDW*, *Kriging*, and *CrossValidation* (Table 3) packages support all those granularities. Below, we focus on the IDW method.

6.5.1. IDW

This method is based on the physical law that everything is related to everything else, but near things are more related than distant things. IDW is one of the simplest and most popular interpolation techniques (Li and Heap, 2008).

The IDW method is very effective when applied to precipitation data, under the condition that the optimal exponent (p) is identified (Lu and Wong, 2008), (Li et al., 2011), (Keblouti et al., 2012), (Chen and Liu, 2012). It is possible to determine the optimal value of p minimizing the Root Mean Squared Error (RMSE).

The *IDW* package allows to interpolate the measured data (and collected in the table `precipitation`), adopting the four levels of granularity mentioned in Section 4. For example, via the UDF `idw_whithinDays()` (*Appendix*), it is possible to apply the IDW method to the values of the precipitation accumulated in an arbitrary number of days. The *CrossValidation* package supports the calculation of the RMSE through four routines referring to the four different temporal granularities supported by the packages of Table 3.

Through the UDF `idw_crossValidation_withinDays()` (*Appendix*), we calculated the RMSE for values of p ranging between 0 and 10, for six distinct daily intervals: 01-05, 06-10, 11-15, 16-20, 21-25 and 26-30 of November 2007. The numerical results are collected in Table 5. It shows that values of p between 1 and 3 return a very low error for each of the six time intervals considered. This result is in line with the findings of studies already conducted. For instance, Chen and Liu (2012) found that the optimal exponent value varies between 0 and 5; Ahani et al. (2012) report a case study on precipitation in Iran for a period of 33 years for which they found that the optimum value of p is 2.

Set $p = 2$, $nx = ny = 300$ (to know about nx and ny please refer to the *Appendix*), we got (by using `idw_whithinDays()`) the maps of the rainfall over Lazio, for the entire month of November 2007 (Figure 8).

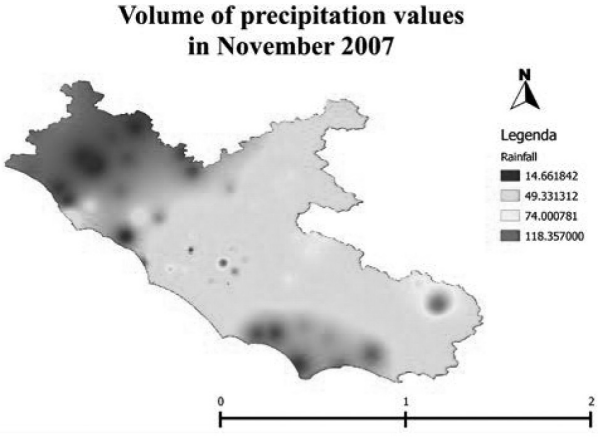

Figure 8. The raster map of the total precipitation volume over Lazio, in November 2007.

Before closing the section, we propose a comparison between the raster map in Figure 8, with that obtained by using QGIS (Figure 9). As can be seen, the two maps are almost identical.

	RMSE					
p	01-05	06-10	11-15	16-20	21-25	26-30
0	5.594	2.474	7.475	4.492	6.878	7.645
1	**5.398**	2.212	7.060	3.449	**6.766**	5.069
2	5.404	1.895	6.572	**2.997**	**6.775**	3.410
3	5.423	**1.803**	**6.458**	3.068	6.926	**3.240**
4	5.446	1.856	6.509	3.230	7.158	3.405
5	5.472	1.919	6.578	3.345	7.393	3.549
6	5.496	1.969	6.637	3.417	7.591	3.639
7	5.516	2.006	6.682	3.465	7.746	3.690
8	5.531	2.034	6.717	3.498	7.868	3.720
9	5.544	2.055	6.742	3.522	7.966	3.739
10	5.555	2.071	6.760	3.539	8.048	3.751

Table 5. The RMSE values for different p values.

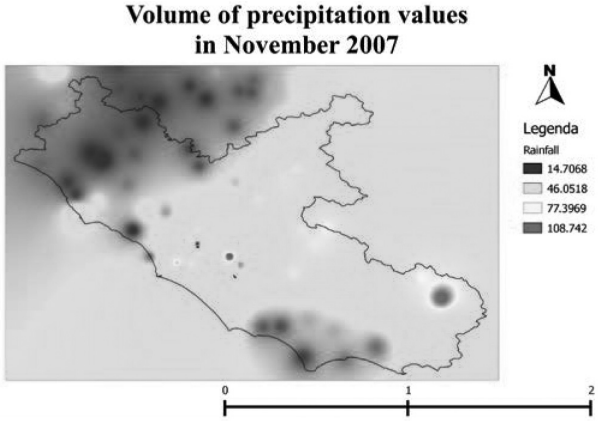

Figure 9. Result of the interpolation via QGIS. Notice that QGIS interpolates over a square region.

7. Conclusions

The adopted solution setting aims at encouraging nontechnical users, after a short on-the-job training, to manipulate by themselves the measurements coming from rain gauges located over a geographical area. The involved stakeholders are only required to familiarize with the objects that the SDB exposes and invoke their execution.

Our vision for the near future hopes that national agencies with the responsibility of land government (in the case of Italy the Department of Civil Protection) will make available periodically (e.g. on an annual basis) an SDB about the historical rainfall data, enhanced with a wide repertoire of objects (*views* and UDFs) oriented to analyse those data. In addition to the SDB, it will be sufficient to provide a user's guide so that anyone who has the desire or the need can take advantage of this service to perform studies focused on the portions of the territory of his interest.

Last but not least, it is worthwhile to point out that the solution framework we have implemented to carry out the experience reported in the paper can be replicated in other areas of high social impact that involve large spatio-temporal data, keeping the operational effectiveness unaltered.

References

[1] H. AHANI ET AL., An investigation of trends in precipitation volume for the last three decades in different regions of Fars province, Iran. *Theoretical and Applied Climatology*, **109** (2012), 361–382.

[2] P. A. BOSTAN, G. B. M. HEUVELINK, S. Z. AKYUREK, Comparison of regression and kriging techniques for mapping the average annual precipitation in Turkey. *International Journal of Applied Earth Observation and Geoinformation*, **19** (2012), 115–126.

[3] D. J. BURNETTE, D. W. STAHLE, Computer assisted screening, correction, and analysis of historical weather measurements. *Computers & Geosciences*, **54** (2013), 309–317.

[4] A. J. CANNON, Quantile regression neural networks: Implementation in R and application to precipitation downscaling, *Computers & Geosciences*, **37** (2011), 1277–1284.

[5] C. J. CARLETON, R. A. DAHLGREN, K. W. TATE, A relational database for the monitoring and analysis of watershed hydrologic functions: I. Database design and pertinent queries. *Computers & Geosciences*, **31** (2005), 393–402.

[6] F. W. CHEN, C.-W. LIU, Estimation of the spatial rainfall distribution using inverse distance weighting (IDW) in the middle of Taiwan. *Paddy and Water Environment*, **10** (2012), 209–222.

[7] U. HABERLANDT, Geostatistical interpolation of hourly precipitation from rain gauges and radar for a large-scale extreme rainfall event. *Journal of Hydrology*, **332** (2007), 144–157.

[8] T. HENGL, *A Practical Guide to Geostatistical Mapping*. Office for Official Publications of the European Communities, Luxembourg, 2009.

[9] M. KEBLOUTI, L. OUERDACHI, H. BOUTAGHANE, Spatial Interpolation of Annual Precipitation in Annaba-Algeria – Comparison and Evaluation of Methods. *Energy Procedia*, **18** (2012), 468–475.

[10] J. LI, A. D. HEAP, A Review of Spatial Interpolation Methods for Environmental Scientists. *Geo-science Australia Record*, **23** (2008), 137.

[11] Q. LI, Y. CHEN, Y. SHEN, X. LI, J. XU, Spatial and temporal trends of climate change in Xinjiang, China. *Journal of Geographical Science*, **21**(6) (2011), 1007–1018.

[12] G. Y. LU, D. W. WONG, An adaptive inverse-distance weighting spatial interpolation technique. *Computers & Geosciences*, **34** (2008), 1044–1055.

[13] E. J. PEBESMA, Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, **30** (2004), 683–691.

[14] J. POKORNY, Database architectures: Current trends and their relationships to environmental data management. *Environmental Modelling & Software*, **21** (2006), 1579–1586.

[15] P. ROWHANIA, D. B. LOBELL, M. LINDERMANC, N. RAMANKUTTYA, Climate variability and crop production in Tanzania. *Agricultural and Forest Meteorology*, **151** (2011), 449–460.

[16] J. SHEN, A. PARKER, J. RIVERSON, A new approach for a Windows-based watershed modeling system based on a database-supporting architecture. *Environmental Modelling & Software*, **20** (2005), 1127–1138.

[17] S. STEINIGER, E. BOCHER, An Overview on Current Free and 61 Open Source Desktop GIS Developments. *International Journal of Geographical Information Science*, **23** (2009), 1345–1370.

[18] H. TABARI, P. H. TALAEE, Temporal variability of precipitation over Iran: 1966–2005. *Journal of Hydrology*, **396** (2011), 313–320.

[19] M. O. WARD, W. L. POWER, P. KETELAAR, A computational environment for the management, processing, and analysis of geological data. *Computers & Geosciences*, (1996), 1123–1131.

Received: January, 2014
Revised: May, 2014
Accepted: June, 2014

Contact addresses:
Paolino Di Felice
Department of Ingegneria Industriale e dell'Informazione ed Economia
University of L'Aquila
Italy
e-mail: paolino.difelice@univaq.it

Luca Finocchio
Daniele Leombruni
Vittoriano Muttillo
Master students in Engineering Informatics
University of L'Aquila
Italy

PAOLINO DI FELICE is a professor of computer science at the Department of Industrial and Information Engineering & Economics of the University of L'Aquila, since 1999. He has authored or co-authored about 100 articles in international journals, books, and conference proceedings in the areas of programming methodologies, relational and spatial databases. He has also carried out a consistent activity of technological transfer in collaboration with several IT firms. His research has been funded by national and international institutions and carried out in collaboration with researchers from several countries (Italy, Holland, Germany, and the USA). He has served within national and international committees. For the last few years, he has been a regular member of the program committee of the International Conference on Integrated Information (IC-ININFO).

LUCA FINOCCHIO, DANIELE LEOMBRUNI AND VITTORIANO MUTTILLO are Master students of engineering informatics at the University of L'Aquila (Italy).

Appendix

The SQL/DDL statements to create the four tables of the SDB (Section 3).

```
CREATE TABLE rain_gauge (
 id_gauge     integer PRIMARY KEY,
 name         character varying(50),
 num_station  integer,
 location     geometry);

CREATE TABLE precipitation (
 gauge  integer,
 date   date,
 time   time,
 accumulated double precision,
 PRIMARY KEY (gauge, date, time),
 FOREIGN KEY (gauge) REFERENCES Rain_gauge(id_gauge)
        ON DELETE NO ACTION ON UPDATE CASCADE );

CREATE TABLE region (
 id_region  integer PRIMARY KEY,
 name       character varying(50),
 boundary   geometry );

CREATE TABLE district (
 id_district  integer PRIMARY KEY,
 region       integer,
 name         character varying(50),
 boundary     geometry,
FOREIGN KEY (region) REFERENCES Region(id_region)
              ON DELETE RESTRICT ON UPDATE CASCADE );
```

The remainder of this appendix collects the code of some of the UDFs mentioned in Section 5.

The UDF `import_precipitation()`.

It migrates the data about the rain measurements into table `precipitation`.

Inputs:

- `path(text)`: path to reach the file of the measurements data
- `date(text)`: month to which the measurements refer (date format: `yyyy-mm`).

```
CREATE OR REPLACE FUNCTION import_precipitation(path text, date text)
RETURNS void AS
$BODY$
  INSERT INTO precipitation (gauge, date, time, accumulated)
    SELECT  gauge, date, time, accumulated
    FROM    UDF_0(path, date)
            t(gauge integer, time time, station text,
            accumulated double precision, date date)
$BODY$
LANGUAGE SQL
```

The (PL/R) *UDF_0* returns a set of records, each composed of five values, without specifying for them a strucure in the usual relational sense. The command `t(...)` allows to assign to each value within the generic record a "header", specifying for each column the *name* and the *data type*.

UDF_1

It calculates the sums of the progressive accumulated rain every 15 minutes in a given rain gauge in a predetermined interval of hours of a specific day.

Inputs:

- `gauge_name` (`text`): name of the rain gauge
- `day` (`text`): the day of interest
- `from_h` (`text`): the initial hour
- `to_h` (`text`): the final hour

Output:

A set of tuples of type `rainfall`.

`rainfall` is a user data type defined as follows:

```
 CREATE TYPE rainfall AS (
  date  date,
  time  time without time zone,
  sum   double precision )
```

The PL/pgSQL code:

```
CREATE OR REPLACE FUNCTION UDF_1(gauge_name text, day text, from_h text, to_h text)
RETURNS SETOF rainfall AS
$BODY$
DECLARE
    row     precipitation%rowtype;
    result  rainfall;
BEGIN
    result.sum=0;
    FOR row IN
            SELECT  *
            FROM    precipitation AS d, rain_gauge AS r
            WHERE   r.id_gauge=d.gauge AND r.name=gauge_name AND
                    d.date=CAST(day as date) AND
                    d.time BETWEEN CAST(from_h as time) AND CAST(to_h as time)
            ORDER BY d.time
    LOOP
            result.date=row.date; result.time=row.time;
            IF row.accumulated IS NOT NULL THEN
                  result.sum=result.sum+row.accumulated;
            ELSE
                  result.sum=result.sum;
            END IF;
            RETURN NEXT result;
    END LOOP;
    RETURN;
END
$BODY$
LANGUAGE 'plpgsql'
```

UDF_7

It returns a .pdf file containing the chart with the hours on the *x*-axis and the amount of rainfall measured (mm) on the *y*-axis.

Inputs:

- `path (text)`: path of the (.pdf) file to be created
- `gauge_name (text)`: name of the pluviometer of interest
- `x1 (time[])`: array about the time of sampling
- `y (double precision[])`: array of the measurements

Output:

A boolean value: `true` if the pdf file has been created, `false` otherwise

The PL/R code:

```
CREATE OR REPLACE FUNCTION UDF_7(path text, gauge_name text, date text,
               x1 time without time zone[], y double precision[])
RETURNS boolean AS
$BODY$
   x2 <- strptime(x1, format='%H:%M');
   x <- as.double(x2);
   l <- length(x);
   pdf(path);
   plot(x, y, xlab='Time (LT) [hh-mm-ss]', ylab='Rainfall [mm]',
   main=paste('Rainfall at', gauge_name, ' rain gauge on ', date), xaxt='n');
   axis(1, at=c(x[1], x[l/2],x[l]), labels=c(x1[1], x1[l/2], x1[l]));
   lines(x, y, col='blue', lwd=2);
   legend(x[1],max(y),c('Rainfall Value','Zigzag Curve'),lty=c(-1,1),pch=c(1,-1),
   col=c('black', 'blue'), lwd=c(-1,2), merge=TRUE);
   dev.off();
return (TRUE);
$BODY$
LANGUAGE 'plr';
```

The UDF `plot_withinHours()`

It plots the sum of the rainfall accumulated every 15 minutes in a rain gauge in a range of hours of a given day. The drawing is returned as a .pdf file.

Inputs:

- `path (text)`: path of the (.pdf) file to be created
- `gauge_name (text)`: name of the rain gauge
- `from_h (text)`: the initial hour
- `to_h (text)`: the final hour

Output:

A boolean value: `true` if the pdf file has been created, `false` otherwise

The PL/pgSQL code:

```
CREATE OR REPLACE FUNCTION plot_withinHours(path text, gauge_name text,
                          day text, from_h text, to_h text)
RETURNS boolean AS
$BODY$
DECLARE
    r rainfall;
    x time[];
    y double precision[];
    i integer=1;
BEGIN
    FOR r IN
        SELECT *
        FROM UDF_1(gauge_name, day, from_h, to_h)
        LOOP
            x[i]=r.time; y[i]=r.sum; i=i+1;
        END LOOP;
    RETURN UDF_7(path, gauge_name, x, y);
END
$BODY$
LANGUAGE 'plpgsql';
```

The UDF `idw_withinDays()`

It interpolates, by means of the IDW technique, the rain accumulated in all the rain gauges in a given interval of days within a given region (in our case, either *Lazio* or *Umbria*). A geocoded (.asc) raster file is returned. This function calls four (*Auxiliary*) UDFs. The (PLpgSQL) UDF_4 computes the sum of the precipitation accumulated by all the rain gauges in the given interval of days within the given region. The (PL/R) UDF_14 computes the interpolation grid, given the dimensions and the coordinates x and y of a square. The (PLpgSQL) UDF_15 filters the points of the grid returned by UDF_14 within the given region. The (PL/R) UDF_16 interpolates, by calling the function `idw` of the library *gstat*, the data received as inputs and returns a geocoded (.asc) raster file built by linking the function `raster` of the library *raster*.

Inputs:

- `region_name (text)`: name of the region of interest (either Lazio or Umbria)
- `from_d (text)`: the initial day
- `to_d (text)`: the final day
- `path (text)`: path of the (.pdf) file to be created
- `power (double precision)`: value of the exponent of the IDW method
- `nx (int)`: number of points of the grid along the x-axis
- `ny (int)`: number of points of the grid along the y-axis

Output:

The actual number of points interpolated.

The PL/pgSQL code:

```
CREATE OR REPLACE FUNCTION idw_whithinDays(region_name text, from_d text, to_d text,
                    path text, power double precision, nx integer, ny integer)
RETURNS integer AS
$BODY$
DECLARE
    id_gau integer[];
    gauge_na character varying(50)[];
    acc double precision[];
    row record;
    x double precision[];
    y double precision[];
    i integer;
    row1 record;
    xg double precision[];
    yg double precision[];
    ig integer;
    row2 record;
    xgf double precision[];
    ygf double precision[];
    igf integer;
BEGIN
    i=0;
    FOR row IN
        SELECT  id_gauge AS id, gauge_name AS name, sum AS a,
                      ST_X(location) AS xx, ST_Y(location) AS yy
        FROM    UDF_4(from_d, to_d)
    LOOP
        id_gau[i]=row.id;
        gauge_na[i]=row.name;
        x[i]=row.xx;
        y[i]=row.yy;
        acc[i]=row.a;
        i=i+1;
    END LOOP;

    ig=0;
    FOR row1 IN
        SELECT  *
        FROM    UDF_14(11, 14.5, 40.5, 44, nx, ny)
                t(xgrid double precision, ygrid double precision)
    LOOP
        xg[ig]=row1.xgrid;
        yg[ig]=row1.ygrid;
        ig=ig+1;
    END LOOP;

    igf=0;
    FOR row2 IN
        SELECT  *
        FROM    UDF_15(region_name, xg, yg)
                t(is_in boolean, xgridfilter double precision,
                    ygridfilter double precision)
```

```
    LOOP
        xgf[igf]=row2.xgridfilter;
        ygf[igf]=row2.ygridfilter;
        igf=igf+1;
    END LOOP;

    RETURN UDF_16(id_gau, gauge_na, x, y, xgf, ygf, acc, power, path);
END;
$BODY$
LANGUAGE plpgsql;
```

The UDF `idw_crossValidation_withinDays()`

It performs the cross validation of the IDW method, run by changing the value of the exponent. All the interpolations refer to the same interval of days given as input. The cross validation is coded inside the (PL/R) UDF_17, which calls specific *gstat* functions that carry out the actual computations.

Inputs:

- `from_d (text)`: initial day
- `to_d (text)`: final day
- `from_p (double precision)`: lower bound of the exponent
- `to_p (double precision)`: upper bound of the exponent
- `by_p (double precision)`: step of increment of the exponent

Output:

A set of records about the RMSE for each exponent value.

The PL/pgSQL code:

```
CREATE OR REPLACE FUNCTION idw_crossValidation_withinDays(from_d text, to_d text, from_p
                    double precision, to_p double precision, by_p double precision)
RETURNS SETOF record AS
$BODY$
DECLARE
    id_gau integer[];
    gauge_na character varying(50)[];
    acc double precision[];
    row record;
    x double precision[];
    y double precision[];
    i integer;
    row1 record;
    ret power_RMSE;
BEGIN
    i=0;
    FOR row IN
        SELECT  id_gauge AS id, gauge_name AS name, sum AS a,
                ST_X(location) AS xx, ST_Y(location) AS yy
        FROM    UDF_4(from_d,to_d)
```

```
        LOOP
          id_gau[i]=row.id;
          gauge_na[i]=row.name;
          x[i]=row.xx;
          y[i]=row.yy;
          acc[i]=row.a;
          i=i+1;
        END LOOP;
    FOR row1 IN
        SELECT  *
        FROM    UDF_17(id_gau, gauge_na, x, y, acc, from_p, to_p, by_p)
                    t(idp double precision, val double precision)
        LOOP
          ret.power =row1.idp;
          ret.RMSE =row1.val;
          RETURN next ret;
    END LOOP;
END;
$BODY$
LANGUAGE plpgsql;
```