RDBMS Framework for Contour Identification*

Łukasz Sosnowski[1,2] and Dominik Ślęzak[3,4]

[1] Systems Research Institute, Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland
[2] Dituel Sp. z o.o.
ul. Ostrobramska 101 lok. 206, 04-041 Warsaw, Poland
[3] Institute of Mathematics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
[4] Infobright Inc.
ul. Krzywickiego 34 lok. 219, 02-078 Warsaw, Poland
l.sosnowski@dituel.pl, slezak@infobright.com

Abstract. We discuss practical aspects of implementation of the compound object identification methodology. In this particular paper, we focus on the RDBMS framework used for storing, processing and comparing objects. We consider SQL for modeling segmentation, granulation and computation of similarities between contours extracted from images. The results indicate that, in our case, the best strategy is to combine the SQL-based and Java-based methods of data processing.

Keywords: RDBMS, Image Processing, Comparators.

1 Introduction

This paper continues our research on compound object comparators (cf. [3]). In [8], we described their application in the commercial project aimed at visualization of the results of the 2010 Polish Self-Government Elections.[5] The task was to display the attendance ratios for the administrative areas of Poland. There were three sources of input data: 1. Attendance in every area; 2. Contour of every area; 3. The map of Poland divided onto areas. Attendance results and contours were labeled with the areas' administrative codes. However, the codes were not present at the map. In order to identify the areas extracted from the map, we had to match them against the repository of reference objects.

The above application was originally implemented in Java. In this paper, basing on our previous experiences [6], we investigate its partial re-implementation within the RDBMS environment, where the steps of image processing and contour identification are represented by a series of ROLAP cubes (cf. [2]).

* The second authors was supported by the grant N N516 077837 from the Ministry of Science and Higher Education of the Republic of Poland, and the National Centre for Research and Development (NCBiR) under the grant SP/I/1/77065/10 by strategic scientific research and experimental development program: "Interdisciplinary System for Interactive Scientific and Scientific-Technical Information".

[5] wybory2010.pkw.gov.pl/att/1/eng/000000.html

CONCURRENCY, SPECIFICATION AND PROGRAMMING
M. Szczuka et al. (eds.): Proceedings of the international workshop CS&P 2011
September 28-30, Pułtusk, Poland, pp. 487-498

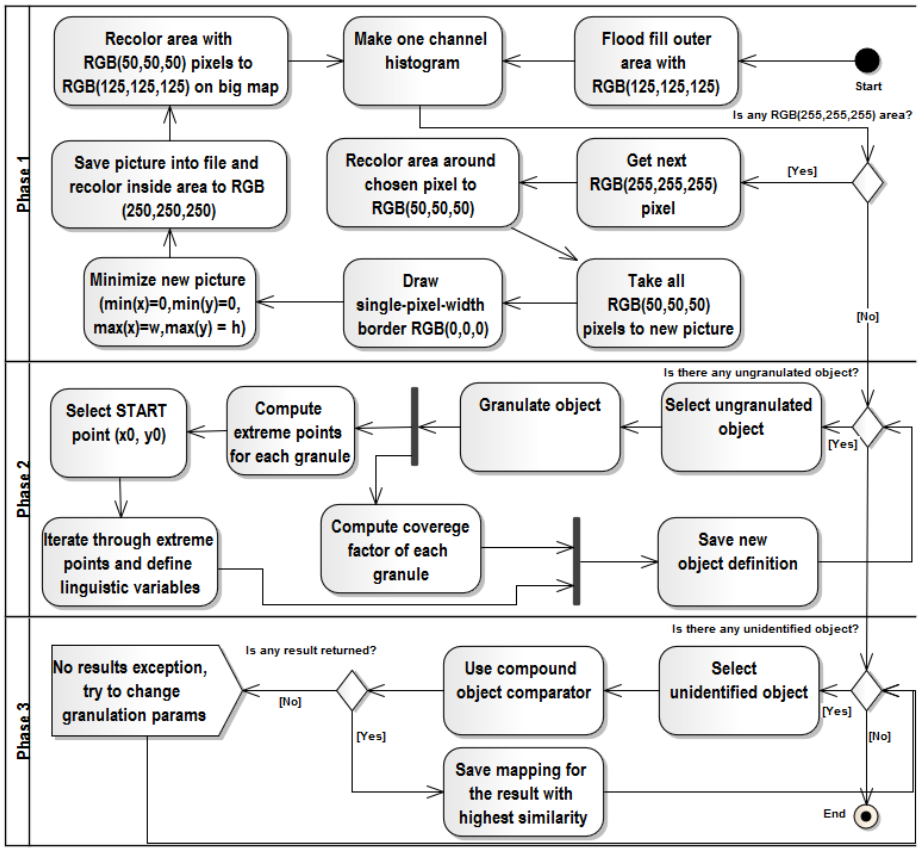

Fig. 1. Activity diagram for the contour identification method introduced in [8].

The paper is organized as follows: In Section 2, we recall the overall approach. In Section 3, we interpret it by means of a ROLAP-based data flow. In Section 4, we discuss possible usage of the RDBMS framework. In Section 5, we introduce the architecture that includes the methods implemented in both SQL and Java. Results and conclusions can be found in Sections 6 and 7, respectively.

2 Comparator-based Identification Algorithm

The algorithm introduced in [8] consists of the three phases: segmentation, granulation, and identification (see Figure 1). Firstly, we acquire objects, i.e. administrative areas from the map of Poland. Secondly, we extract granules of objects (cf. [4]), compute their characteristics, and synthesize them into granular descriptions of objects. We consider two types of characteristics, although the framework is open for others. In case of the first type, referred as *coverage*, we use the histogram technique [5] to compute a vector of overlaps of image's gran-

ules with an area that the image presents. In case of the second type, *contour*, we produce linguistic description (cf. [3]) of directions of lines connecting the extremum points of the area's contour within each of granules. In the third phase, we compute similarities between the input and reference objects with respect to each of types of granular descriptions, and we synthesize the similarity scores. For *coverage*, similarity is opposite to the sum of overlap differences. For *contour*, we employ the Levenshtein's edit distance between linguistic descriptions.

3 Data Flow in ROLAP Cubes

The ROLAP cubes store data about objects at various levels of aggregation [2]. The idea is to design and connect with each other the cubes corresponding to the phases outlined in Section 2. The cubes can be fed with the data sequentially. Sub-tasks such as, e.g., the *coverage* and *contour* computation can be parallelized. The scheme of cube feeding can be seen in Figure 2.

Each aggregation changes the level of data granularity. ROLAP cubes allow for fast data access at the previously specified levels. In particular, we can preprocess all reference objects and store them conveniently in various aggregated forms. If it is necessary to step down to more detailed data, we can do it using SQL. Thus, one should consider RDBMS solutions enabling efficient storage of massive amounts of both original and processed data, as well as fast execution of compound aggregations (see e.g. [1] and [7] for references).

The cubes in Figure 2 have some common dimensions. They create a constellation (see Figure 3). Common dimensions are useful for various types of computations. The constellation follows the uniform ontology of attributes and their values. It is the storage layer of our application, separated from the layer of algorithms that produce and use data. In this section, we describe each of cubes in more detail. We go back to the algorithmic layer in Section 4.

The Cr cube contains atomic information about images. Each pixel of each image corresponds to a single row in the table `fact_images`. We refer to [6] for the detailed description of Cr; thus, we do not include it into Figure 3. Although application considered in [6] was quite different from contour identification, the form of Cr can be universal for various tasks of image processing.

The Gc cube (*Granules cube*) reflects the beginning of the granulation phase. The corresponding table `fact_granule_pixels` contains rows representing pixels belonging to the image borders. Additional columns indicate membership of pixels into granules obtained by partitioning images according to predefined resolution parameter. If there is no need to change resolution, all subsequent phases can use Gc instead of Cr. Otherwise, Cr needs to be recalculated.

The Ce cube (*Extrema cube*) describes only those image border pixels, which were detected as having the extremum coordinates, in each of granules locally. We search for extremum values for both coordinates (keeping the other coordinate locally maximized). It generates up to four extremum pixels per granule. The detailed equations for such extremum pixels can be found in [8].

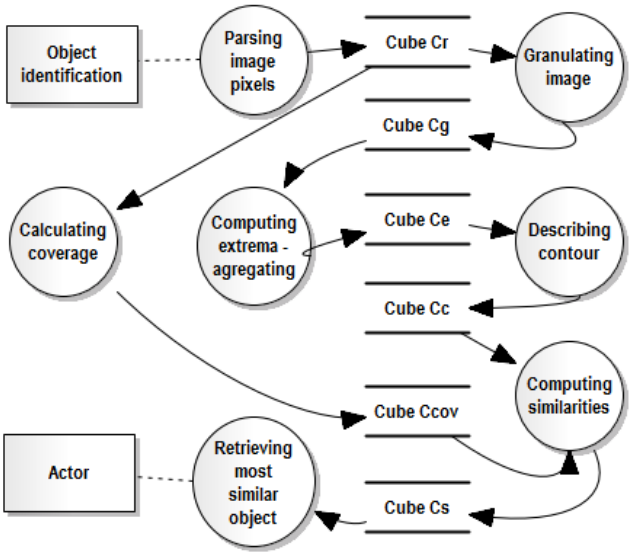

Fig. 2. Data flow diagram of the cube feeding. The following abbreviations are used: Cr stands for `fact_images`; Cg – `fact_granule_pixels`; Ce – `fact_granule_extrema`; Cc – `fact_contours`; Ccov – `fact_area_coverage`; Cs – `fact_object_similarities`.

The Cc cube (*Contour cube*) stores linguistic descriptions of contours. The corresponding table `fact_contours` includes the sequences of letters such as *L* for *left*, *R* for *right*, etc., describing directions of lines connecting extremum pixels. There are two possibilities to represent the linguistic descriptions in a cube: as single VARCHAR values or, after a modification of `fact_contours`, as pairs of extremum values and their single-letter characteristics.

The Ccov cube (*Coverage cube*) stores information about overlaps of the granules with the areas. Each image is described by a sequence of rows corresponding to the granules labeled with the coverage factor. Ccov is analogous to one of the ROLAP cubes considered in [6]. The difference lays in one of dimensions: `d_granules`. Here, it may represent the granules of arbitrary objects, while in our previous research it was dedicated to images only.

The Cs cube (*Similarity cube*) stores the outcomes of comparators, i.e. the similarity scores for the pairs of objects. The scores are stored in order to avoid repeating computations under similar parameters. Each tuple is labeled with information about granulation resolution applied to build the score. Thus, Cs can store multiple results obtained for the same pair of objects.

As a summary, we can process objects using various settings and store the obtained intermediate results. For entirely new objects or parameters, we can execute SQL on the Cr cube. We can also extract specific data pieces and continue calculations outside RDBMS. In both scenarios, it is important to operate with a clear data schema such as the one illustrated by Figure 3.

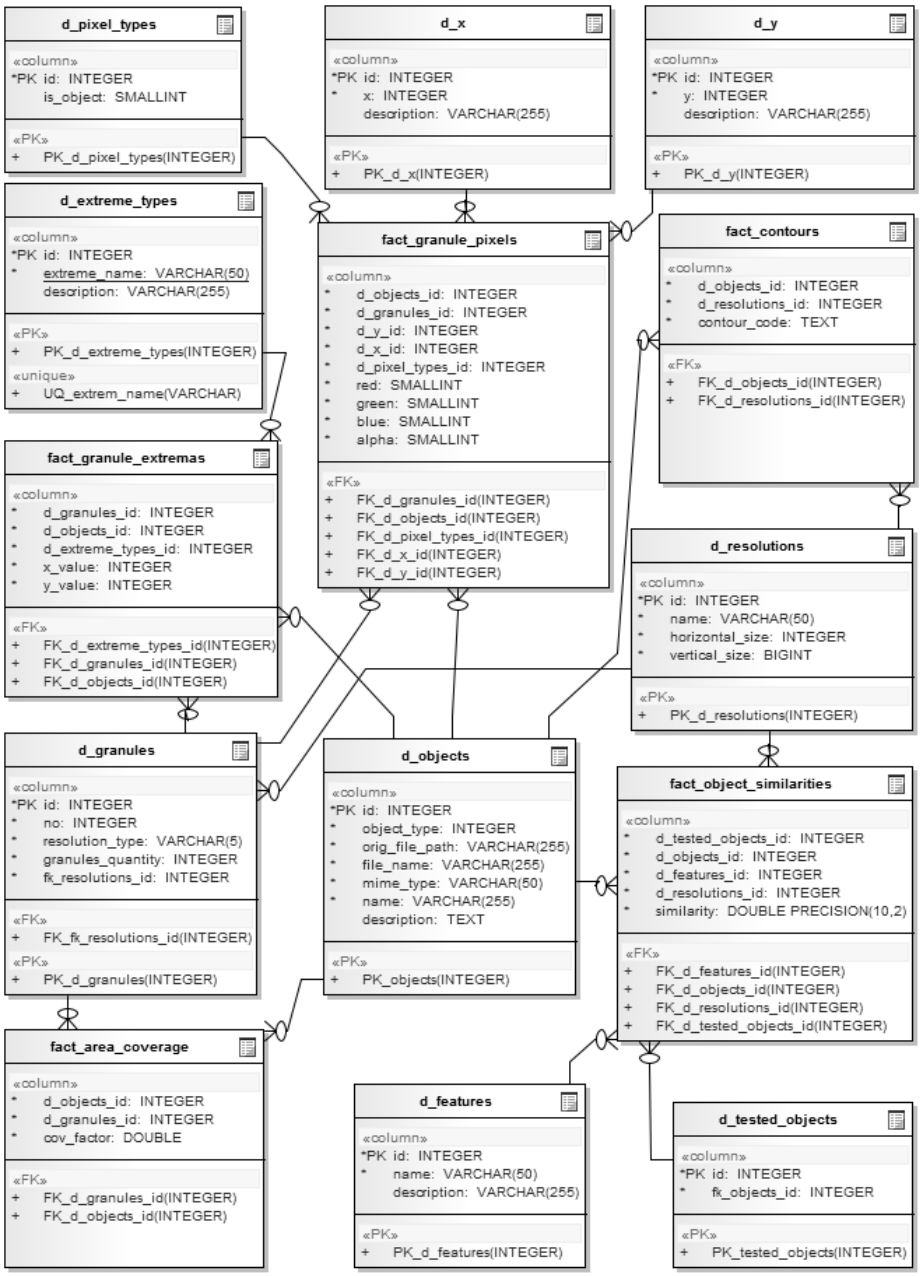

Fig. 3. Constellation of ROLAP cubes designed for the contour identification problem. Fact tables and dimension tables are denoted by prefixes `fact_` and `d_`, respectively.

4 Implementation Ideas

4.1 Phase 1: Segmentation

Segmentation is the process of acquiring objects from a bigger object. The detected objects can be saved as separate image files or, in case of the RDBMS approach, stored by means of rows-pixels with appropriate object identifiers. We propose to store objects in a relational form. However, their extraction and further analysis can be designed both inside and outside RDBMS.

There are numerous methods of object detection. For instance, we may look at pixels at a given brightness level basing on the image histogram [5]. Starting from them, we may use the *flood fill*[6] algorithm based on 4-connected neighborhood[7] in order to repaint the areas bounded by contour borders.

Implementation of *flood fill* within RDBMS may be difficult. In the database script languages there is often a bound for the number of recursion levels. There may be also problems with poor performance, e.g., if the repainting operation is implemented by means of data updates. One may overcome this obstacle by using temporary tables initialized as the upper approximations of the areas to be extracted. In some RDBMS solutions, we may store such approximations as in-memory tables. On the other hand, we may also think about the segmentation phase as a typical ETL[8] task to be executed outside RDBMS.

Regardless of specific implementation, *flood fill* is supposed to repaint the area's interior using a different color. Then we read the maximum vertical and horizontal ranges of repainted pixels, add two pixels of a margin and create the relational form of the obtained image. Generation of the area's borders is based again on the analysis of 4-connected neighborhoods. We can examine pixels iteratively and repaint the resulting borders using one more color.

4.2 Phase 2: Granulation and Factor Computation

Granulation is executed against the Cr cube. It aims at splitting objects onto smaller pieces. Our approach requires specification of two parameters, m and n, which yield $m \times n$ granules for each area. First, we need to compute the lengths of intervals qualifying pixels to particular granules. Figure 4 illustrates an example of SQL procedure (referred as `GRANULATION PROCEDURE`) resulting in the granulated objects. The computed interval lengths are denoted as wl and hl. Granules are identified by their vertical and horizontal ordinal coordinates. Each pixel of each area is labeled with its granule's identifiers.

The next step is to calculate the *coverage* characteristics for all granules of all images. We refer to [8] for more detailed discussion on how to extract and use them. Figure 4 includes also SQL statements responsible for the *coverage* extraction and the *coverage*-based similarity computations.

[6] `en.wikipedia.org/wiki/Flood_fill`
[7] `en.wikipedia.org/wiki/Pixel_connectivity`
[8] `en.wikipedia.org/wiki/Extract,_transform,_load`

```
##GRANULATING PROCEDURE
CREATE PROCEDURE granulateObjects(IN m INTEGER,IN n INTEGER)
BEGIN
SELECT d_objects_id,g.id as d_granules_id,y,x,
case when red=50 and green=50 and blue=50 then 0 else 1 end as d_pixel_types_id,
red,green,blue,alpha FROM(
    SELECT fi.d_objects_id,d_x_id as x,d_y_id as y,
    case when d_x_id<>0 and d_x_id mod r.wl =0 then d_x_id/r.wl
    else (d_x_id-(d_x_id mod r.wl))/r.wl +1 end as xg,
    case when d_y_id<>0 and d_y_id mod r.hl =0 then d_y_id/r.hl
    else (d_y_id-(d_y_id mod r.hl))/r.hl +1 end as yg,
    red,green,blue,alpha
    FROM fact_images fi
    INNER JOIN tmp_wl_hl r ON r.d_objects_id=fi.d_objects_id
    WHERE RED=0 AND GREEN=0 AND BLUE=0)tmp
INNER JOIN d_granules g
on cast((yg-1)*m + xg as SIGNED)=g.no and g.g_x=tmp.xg and g.g_y=tmp.yg
INNER JOIN d_resolutions rs on g.d_resolutions_id=rs.id WHERE
rs.horizontal_size=m and rs.vertical_size=n INTO OUTFILE 'K:/data/granules.csv';
END//

CREATE PROCEDURE prepareCoverage(m INTEGER)
BEGIN
select d_objects_id,g.id ,coverage/g_all as factor FROM(
SELECT fi.d_objects_id,
case when d_x_id<>0 and d_x_id mod r.wl =0 then d_x_id/r.wl
else (d_x_id-(d_x_id mod r.wl))/r.wl +1 end xg,
case when d_y_id<>0 and d_y_id mod r.hl =0 then d_y_id/r.hl
else (d_y_id-(d_y_id mod r.hl))/r.hl +1 end as yg,
count(*) as coverage,r.wl*r.hl as g_all
 from (SELECT fi.d_objects_id,d_x_id,d_y_id FROM fact_images fi
 WHERE RED=250 AND GREEN=250 AND BLUE=250 ) fi
 INNER JOIN tmp_wl_hl r ON r.d_objects_id=fi.d_objects_id
 GROUP BY fi.d_objects_id,
 case when d_x_id<>0 and d_x_id mod r.wl =0 then d_x_id/r.wl
 else (d_x_id-(d_x_id mod r.wl))/r.wl +1 end ,
 case when d_y_id<>0 and d_y_id mod r.hl =0 then d_y_id/r.hl
 else (d_y_id-(d_y_id mod r.hl))/r.hl +1 end) p
 INNER JOIN d_granules g on cast((yg-1)*m + xg as SIGNED)=g.no
 and g.g_x=p.xg and g.g_y=yg INTO OUTFILE 'K:/data/tmp_area_coverage.csv';
END//
#CALCULATE COVERAGE MEASURE
select d_objects_id_test,d_objects_id_ref,sum(substract) sum_subst,
granules_quantity from( select  o.id as d_objects_id_test,
o2.id as d_objects_id_ref, abs(test.cov_factor - ref.cov_factor) as substract,
g.granules_quantity from tmp_fact_area_coverage test inner join
d_objects o on  test.d_objects_id = o.id inner join
fact_area_coverage ref on test.d_granules_id=ref.d_granules_id inner join
d_objects o2 on  ref.d_objects_id = o2.id  inner join d_granules g on
ref.d_granules_id=g.id where o.is_reference=0 and o2.is_reference=1) t
group by d_objects_id_test,d_objects_id_ref,granules_quantity
INTO OUTFILE 'K:/data/coverages.csv';
```

Fig. 4. SQL-based implementation: Granulation and the *coverage* handling.

```
CREATE PROCEDURE calculateExtremas()
BEGIN
SELECT gs.d_granules_id,gs.d_objects_id,1 as extreme_type,x,max(gs.d_y_id) as y
from fact_granule_pixels gs inner join ( SELECT d_objects_id,d_granules_id,
min(d_x_id) as x from fact_granule_pixels where red=0 and green=0 and blue=0
group by d_objects_id,d_granules_id) t on gs.d_objects_id=t.d_objects_id and
gs.d_granules_id=t.d_granules_id and gs.d_x_id = t.x where gs.red=0 and
gs.green=0 and gs.blue=0  group by gs.d_objects_id,gs.d_granules_id,x  union all
SELECT gs.d_granules_id,gs.d_objects_id,2 as extreme_type,max(d_x_id)as x,y from
fact_granule_pixels gs inner join ( SELECT d_objects_id,d_granules_id,min(d_y_id)
as y from fact_granule_pixels where red=0 and green=0 and blue=0 group by
d_objects_id,d_granules_id order by d_objects_id,d_granules_id) t on
gs.d_objects_id=t.d_objects_id and gs.d_granules_id=t.d_granules_id and
gs.d_y_id = t.y where gs.red=0 and gs.green=0 and gs.blue=0  group by
gs.d_objects_id,gs.d_granules_id,y union all
SELECT gs.d_granules_id,gs.d_objects_id,3 as extreme_type,x,max(gs.d_y_id) as y
from fact_granule_pixels gs inner join ( SELECT d_objects_id,d_granules_id,
max(d_x_id) as x from fact_granule_pixels where red=0 and green=0 and blue=0
group by d_objects_id,d_granules_id) t on gs.d_objects_id=t.d_objects_id and
gs.d_granules_id=t.d_granules_id and gs.d_x_id = t.x where gs.red=0 and gs.green=0
and gs.blue=0  group by gs.d_objects_id,gs.d_granules_id,x  union all
SELECT gs.d_granules_id,gs.d_objects_id,4 as extreme_type,mx(d_x_id)as x,y from
fact_granule_pixels gs inner join ( SELECT d_objects_id,d_granules_id,max(d_y_id)
as y from fact_granule_pixels where red=0 and green=0 and blue=0 group by
d_objects_id,d_granules_id order by d_objects_id,d_granules_id) t on
gs.d_objects_id=t.d_objects_id and gs.d_granules_id=t.d_granules_id and
gs.d_y_id = t.y  where gs.red=0 and gs.green=0 and gs.blue=0  group by
gs.d_objects_id,gs.d_granules_id,y order by d_objects_id,d_granules_id INTO
OUTFILE 'K:/data/extremas.csv'; END//

CREATE FUNCTION getLinguisticVar(x1 INTEGER,y1 INTEGER,x2 INTEGER, y2 INTEGER)
RETURNS VARCHAR(4) BEGIN
DECLARE spanX INTEGER; DECLARE spanY INTEGER; DECLARE lingVar VARCHAR(4);
SET lingVar=''; IF(x1=x2 and y1=y2) THEN  RETURN lingVar; END IF;
SET spanX = abs(x2 - x1); SET spanY = abs(y2 - y1);
IF spanX > spanY THEN
SET lingVar= CASE WHEN (x2<x1) THEN 'L' ELSE 'R' END;
SET lingVar=  CASE WHEN (spanY<>0 and y2<y1) THEN CONCAT(lingVar,'U') ELSE
CONCAT(lingVar,'D') END;
ELSEIF spanX=spanY THEN SET lingVar=CASE WHEN (x2<x1) THEN CONCAT(lingVar,'LL')
ELSE CONCAT(lingVar,'RR') END; SET lingVar=CASE WHEN (y2<y1) THEN
CONCAT(lingVar,'UU') ELSE CONCAT(lingVar,'DD') END; ELSEIF spanX<spanY THEN
SET lingVar=  CASE WHEN (x2<>0 and x2<x1) THEN CONCAT(lingVar,'L') ELSE
CONCAT(lingVar,'R') END; SET lingVar=  CASE WHEN (y2<y1) THEN CONCAT(lingVar,'U')
ELSE CONCAT(lingVar,'D') END; END IF; return lingVar; END//

CREATE PROCEDURE computeContourSimilarity()
BEGIN select * from(select d_objects_id_test,d_objects_id_ref,d_features_id,
d_resolutions_id,memberschipContour(contour_code_test ,contour_code_ref)
as similarity  from tmp_contour_similarity ) a order by d_objects_id_test,
similarity desc INTO OUTFILE 'K:/contour_similarity.csv';
END//
```

Fig. 5. SQL-based implementation: Three steps of the *contour* handling.

In parallel to the above, we compute extremum values for each granule. They are important to create simplified contour representations. It is important to note that extremum values of all granules, for all images, can be extracted using a single SQL statement (see Figure 5). In such situations, the RDBMS framework is of course very efficient, especially if the chosen database engine has good performance characteristics for aggregate queries.

Now, the algorithm needs to find a pixel to start describing the simplified contour. The corresponding query should work on the Ce cube. For each image simultaneously, it computes a pixel with the minimum X coordinate. For each image, we then analyze directions of lines connecting the extremum pixels. Such linguistic description can include combinations of directions. Figure 5 illustrates also this part (the function `getLinguisticVar`).

4.3 Phase 3: Similarity Computation

The overall goal of the third phase is to appoint the most similar reference object to each of the inputs. Such identification is processed by sorting objects relative to their similarities with respect to particular features. The final decision is taken on the basis of interim results and a method of their aggregation.

The features – in our case these are *contour* and *coverage* – are processed by dedicated comparators. An example of SQL computation of the *contour*-based similarity based on the Levenshtein distance is shown in Figure 5 too. In parallel, we can easily encode calculation of the *coverage*-based similarity [8]. Once all results are gathered in the Cs cube, we can proceed with aggregation. Out of many possibilities, we have chosen the mean of similarity scores.

4.4 Additional Re-identification Phase

In case of a larger set of inputs, if the correspondence of the input objects and the reference objects is supposed to be one-to-one, we can apply an additional procedure briefly illustrated by Figure 6. Whenever a single reference object is chosen more than once, we keep its assignment for the input object with the highest aggregated similarity value and disable it for the other inputs. The input objects that lose their assignments need to be identified again, using the pool of yet unassigned reference objects. The procedure is repeated until there are no reference objects assigned to multiple inputs. The time necessary to run this step can be neglected, as there is no need to recompute similarities.

5 Proposed Overall Architecture

Given some initial investigations related to the methods described in Section 4, as well as our previous research reported in [8], we would like to propose the framework designed to process data in two ways: centralized and distributed. In the centralized case, all computations are implemented within the RDBMS environment. In the distributed case, we perform some tasks outside the database,

	ref1	ref2	ref3	ref4	ref5
obj1	**0,91**	0,65	0,23	0,45	0,49
obj2	**0,82**	0,81	0,34	0,42	0,12
obj3	0,23	0,12	**0,88**	0,23	0,31
obj4	0,51	0,43	0,32	**0,94**	0,12
obj5	0,49	0,21	0,14	0,39	**0,89**

	ref1	ref2	ref3	ref4	ref5
obj1	**0,91**	0,65	0,23	0,45	0,49
obj2		**0,81**	0,34	0,42	0,12
obj3	0,23	0,12	**0,88**	0,23	0,31
obj4	0,51	0,43	0,32	**0,94**	0,12
obj5	0,49	0,21	0,14	0,39	**0,89**

Fig. 6. An additional step of the object re-identification.

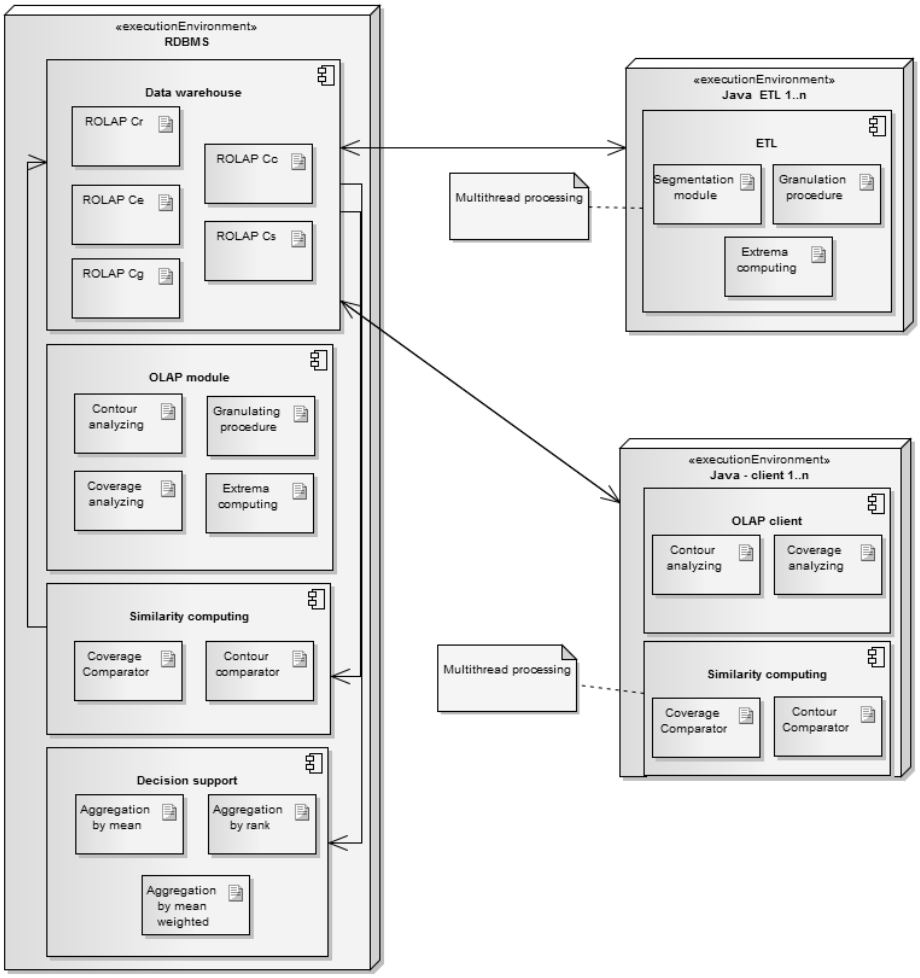

Fig. 7. The proposed overall architecture. The RDBMS and the ETL / OLAP client execution environments correspond to the centralized and distributed computation paradigms, respectively.

achieving more flexibility in their parallelization. Certainly, we do not claim that the RDBMS engines are not able to parallelize the SQL statements or their components internally. Nevertheless, dedicating some computational tasks to the external modules can bring us better overall performance.

The decision how to process a given task may depend on many factors, such as the type of data, features, and available hardware. Regardless of such decisions for particular stages, we recommend to store all intermediate results in the previously introduced ROLAP cubes, in order to provide easiness of both accessing the data and modifying the chain of calculations.

Figure 7 presents the deployment diagram of the proposed solution. The part called *Data warehouse* gathers all cubes, establishing the input to the processing tasks that can be scheduled both inside and outside RDBMS. This happens, e.g., for the OLAP-like activities – they can be both centralized (*OLAP module*) and distributed (*OLAP client*). The distributed strategy is useful whenever there are many steps of data creation and consumption, including recursive procedures executed for separate objects. The centralized approach is better for the tasks that can be modeled as massive aggregations over the whole data, especially if the applied RDBMS technology is optimized for the analytic purposes.

6 Performance Results

We implemented our framework using the ICE (Infobright Community Edition[9]) RDBMS engine [7] and Java 1.6. We report the results obtained on a laptop, T9600 Intel Core Duo, 8GB RAM, 500GB 5400 RPM, Windows 7 64 bit. The tests were divided into three parts, like in the original approach [8].

We decided to design the first phase as an entirely external, Java-based ETL process. Although we attempted to compare it with the ICE-based implementation by choosing the MEMORY engine,[10] the issues raised in Subsection 4.1 turned out prohibitive for such a straightforward RDBMS solution.

The situation is quite different in case of the second phase. As mentioned in Subsection 4.2 (see also Figure 4), the operations of granulation and extrema computation can be modeled by a single massive aggregate SQL statement, executed against all images in the same time. As reported in Table 1, it immediately brings the advantage to the centralized RDBMS-based strategy.

As for the third phase, it depends on the feature that a comparator is based on. While *coverage* is relatively easy to express by an SQL aggregation (see Figure 4), *contour* has a little chance for efficient execution in case of such RDBMS technologies as ICE. Table 1 confirms it. An interesting task for the future would be to find out whether there is a way to approximate the Levenshtein distance by some functions that are easier to model in standard SQL.

The steps related to aggregation of the similarity scores and improvements discussed in Subsection 3.4 require handling very small data, so their specific implementation is not so crucial for the overall performance.

[9] `www.infobright.org`

[10] `dev.mysql.com/doc/refman/5.1/en/memory-storage-engine.html`

Table 1. Phases 2 (granulation, including extraction of extrememum values) and 3 (two types of comparators: *contour* and *coverage*). The implementation versions – Java 1.6 (distributed approach) and SQL (centralized approach) – are marked in brackets.

# of objects	Granulation (Java)	Granulation (SQL)	*Coverage* (Java)	*Coverage* (SQL)	*Contour* (Java)	*Contour* (SQL)
1	1311 ms	30 ms	19 ms	30 ms	90 ms	140 ms
10	1424 ms	50 ms	433 ms	140 ms	219 ms	9030 ms
400	4366 ms	1300 ms	903 ms	5790 ms	6724 ms	279450 ms
2000	19786 ms	4530 ms	13100 ms	19980 ms	80415 ms	>10 min
10000	60185 ms	21790 ms	19000 ms	85920 ms	>10 min	>10 min

7 Conclusions

The obtained results show that, in case of application considered in this paper, there are advantages of both RDBMS-based and Java-based approaches. One of the greatest advantages of RDBMS is a convenient access to data at various levels of aggregation. It is important in case of any changes (such as, e.g., adding new similarity features). On the other hand, Java provides a deeper control on the advanced data structures and multithread processing.

This justifies our proposal of a hybrid architecture for compound object identification, with the data storage managed entirely by RDBMS and with the data processing operations ready to execute in both SQL and Java. The discussion in Sections 5 and 6 gives some insight into how to choose between these two strategies in particular situations. It also draws the directions for future improvements of the overall performance and scalability of our system.

References

1. Agrawal, R., et al.: The Claremont report on database research. SIGMOD Rec. 37(3): 9-19 (2008)
2. Garcia-Molina, H., Ullman, J., Widom, J.: Database Systems: The Complete Book. Prentice-Hall (2008)
3. Kacprzyk, J.: Multistage Fuzzy Control: A Model-Based Approach to Fuzzy Control and Decision Making. Wiley (1997)
4. Pedrycz, W., Kreinovich, V., Skowron, A. (Eds.): Handbook of Granular Computing. Wiley (2008)
5. Russ, J.: The Image Processing Handbook (the 5th Edition). CRC Press (2007)
6. Ślęzak, D., Sosnowski, Ł.: SQL-Based Compound Object Comparators: A Case Study of Images Stored in ICE. In: Proc. of ASEA, CCIS 117, Springer (2010) pp. 304-317
7. Ślęzak, D., Wróblewski, J., Eastwood, V., Synak, P.: Brighthouse: An Analytic Data Warehouse for Ad-hocQueries. PVLDB 1(2): 1337-1345 (2008)
8. Sosnowski, Ł., Ślęzak, D.: Comparators for Compound Object Identification. In: Proc. of RSFDGrC, LNAI 6743, Springer (2011) pp. 342-349