RDBMS Model for Scientific Articles Analytics

Marcin Kowalski[1], Dominik Ślęzak[1,2], Krzysztof Stencel[1],
Przemysław Pardel[3,1], Marek Grzegorowski[1], and Michał Kijowski[1]

[1] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
[2] Infobright Inc.
ul. Krzywickiego 34 lok. 219, 02-078 Warsaw, Poland
[3] Chair of Computer Science, University of Rzeszów
ul. Rejtana 16A, 35-310 Rzeszów, Poland

Abstract. We present the relational database schema aimed at efficient storage and querying parsed scientific articles, as well as entities corresponding to researchers, institutions, scientific areas, et cetera. An important requirement in front of the proposed model is to operate with various types of entities, but with no increase of schema's complexity. Another aspect is to store detailed information about parsed articles in order to conduct advanced analytics in combination with the domain knowledge about scientific topics, by means of standard SQL and RDBMS management. The overall goal is to enable offline, possibly incremental computation of semantic indexes supporting end users via other modules, optimized for fast search and not necessarily for fast analytics, as well as direct ad-hoc SQL access by the most advanced users.

Keywords: RDBMS systems, database schema optimization, SQL-based analytics, document repositories, information retrieval and synthesis.

1 Introduction

One of the major functionalities of the "Interdisciplinary System for Interactive Scientific and Scientific-Technical Information" – a scientific project financed by Polish Government in 2010-2013[1] – is to analyze and intelligently index large collections of scientific articles [12]. From technical point of view, it requires fast access to heterogeneous sources of articles, as well as convenient means for storing information retrieved from those articles. Such tasks may be partially compared to building and using various types of indexes in the Web search and enterprise search solutions. However, one should note that building truly meaningful indexes and other, more compound types of information may require processing and comparing huge sets of articles. Also, the level of detail while accessing and comparing those articles may be crucial for the quality of results.

During initial investigation, as in many other real-life projects, we quickly realized that there might be no single database/repository methodology perfectly

[1] http://ismis2011.ii.pw.edu.pl/post_conference_event.php

R. Bembenik et al. (Eds.): Intelligent Tools for Building a Scient. Info. Plat., SCI 390, pp. 49–60.
springerlink.com © Springer-Verlag Berlin Heidelberg 2012

addressing all of the above expectations [1]. For example, dedicated document stores seem to be the best ones for gathering and managing original files and metadata of articles acquired from different sources, with an additional option of applying structural retrieval / OCR algorithms to their particular *instances*. On the other hand, standard RDBMS engines or key-value stores are appropriate for providing results of the indexing and analytic processes to end users and external tools aiming at search and visualization of the scientific contents. Going further, more analytic-oriented RDBMS solutions are a better choice for managing information about articles and other related entities in a form of data tables that can be efficiently queried using aggregate, nested, quite often ad-hoc SQL statements generated by advanced users or generated automatically by the data mining algorithms aimed at ranking, grouping, et cetera.

In this paper, we concentrate on the last of above aspects, particularly, a relational model for information about articles and related entities that may be appropriate for compound SQL-based analytics. At the beginning, our intension was to store in such model only relatively high-level information and metadata available for articles, in combination with the history of system's usage and the domain knowledge about scientific areas – everything expressed within a unified relational database schema. Actually, we could see similar attempts also in other projects [7], which provided us with additional inspiration and technical hints. However, encouraged by some recent examples of data warehouses storing far more detailed information about compound entities, such as natural language [8] and multimedia [15], we decided to enrich our originally planned model by fully parsed texts of articles (and descriptions of other entities).

The paper is organized as follows: Section 2 outlines fundamental requirements for the relational database schema and its integration with other modules in the project. Section 3 contains preliminary analysis of the most significant challenges and their possible solutions. Section 4 describes three major areas of our schema: generic, analytic, and ontological, populated with *instances*, *objects*, and *concepts*, respectively. Section 5 concludes the paper.

2 Requirements

In order to establish fully functional solution, we keep information about articles and their related entities in two forms – document repository optimized for navigation and RDBMS optimized for analytics. This way, we achieve duality of scientific content storage which has important implications for efficiency and scalability of algorithms designed within the project [12]. It is thus important to consider two levels of requirements: the specifics of relational database schema and the way of interaction of RDBMS part of solution with other modules. Let us outline the main requirements at the first level:

– there are three kinds of information to be stored: gained directly from parsed documents, related to domain knowledge delivered by experts or gained from well-known sources, and retrieved by combining two above kinds

- it has to be prepared for noisy and incomplete information (e.g.: missing metadata, different input file formats, partially known article structure)
- large volumes of data to be stored in a form enabling advanced analytics
- multiple *instances* of the same *objects* (articles, scientists, institutes) may occur because algorithms aimed at avoiding such cases prior to loading data into RDBMS may be insufficient
- it should be open for adding new types of information and entities but in a flexible way, i.e., additions should not enforce schema modifications
- it should enable indexing, querying, and generally reasoning about each part (e.g.: abstract, section) of an article separately and, if necessary, combining results in an article's structure-aware way.

As for the second level of requirements, RDBMS part is going to rebuild its content in an offline fashion. The latest, not yet processed articles (we use timestamp to identify such articles) can be parsed cyclically (e.g.: every week) and then loaded to RDBMS in a bulk load style. The results of SQL-based algorithms working on the relational database can be exported cyclically to external layers, to support search capabilities and user interfaces. On the other hand, the whole data stored in RDBMS should be permanently available to advanced users who (when permitted) can generate direct SQL statements and project members working on algorithmic enhancements (often requiring dynamic creation of new intermediate tables or unmaterialized views).

Given the above scenarios, we tend to look at RDBMS solutions with good data compression and/or abilities to distribute data and query workloads. However, we need to remember that standard MPP database solutions [6] or, e.g., extensions towards MapReduce computations [13] may be risky given a need to process truly compound and diversified analytics. More information about applied software can be found in [12]. In this paper, we focus mainly on the relational database schema specification, although we also comment on some performance results obtained by using one of RDBMS engines [16].

3 Preliminaries

3.1 Multiple *Instances* of the Same *Object*

One of the problems that we encountered during the schema design was the existence of potentially many *instances* of the same article in various suppliers' resources (given some analogies to popular search engines, we may call it duplication). Such cases may actually occur also for other types of *objects*, e.g., scientists or institutes. While parsing and loading information about new articles, we retrieve lots of meaningful information about their authors, references, et cetera. However, in order to keep the loading processes as simple and realistic as possible, we do not conduct full checking whether (and to what degree) the retrieved entities are *instances* of *objects* already stored in RDBMS part.

From the system effectiveness and results representation points of view, it is of course more useful to get information about, e.g., a book in general rather than

about its specific PDF or scanned versions. It brings us to the task of recognizing such situations within RDBMS and storing the relevant information at the level of equivalence classes of *instances* corresponding to the same *objects*. As already mentioned, it might be partially implemented online, during data load, however too sophisticated techniques would significantly decrease the load speed. It may also yield biased results as there is an asymmetry of information between the beginning and the end of load. Thus, we should rather implement and trigger the matching (deduplication) mechanisms after each bigger data load. There are surely plenty of possible matching techniques basing on *instances'* metadata or their internal structures (see e.g. [17]). Most of them can be successfully translated into SQL-based analytic processes, additionally supported (if necessary) by accessing the document store counterpart discussed in Section 2.

As a result, we introduce double identification for entities. First, during new publications (and other types of entities or their parts) are added, the new *instances* are created, assigned with the first level identifiers (column `id_instance`). Then, the matching algorithms merge *instances* (or their parts) into *objects*, which are basically the classes of (parts of) *instances*, assigned with the second level identifiers (column `id_object`). All answers for queries submitted to the system by users are assumed to concern the `id_object` level.

3.2 Sources of Information

There may be multiple algorithms for retrieving information from articles and other entities. In order to identify entities obtained using different algorithms, we introduce the *source of information*. Each (version of) algorithm has its corresponding `id_source`. Algorithms's description is available in additional table `SOURCE` comprising of columns `id_source` (INT) and `source_name` (VARCHAR, e.g.: 'wikipedia_eng_20101231' or 'heuristic_X').

3.3 DICTIONARY Data Type

We introduce the DICTIONARY column type, which reflects a kind of internal normalization of relational database schema. Each value of a column of DICTIONARY type is assumed to be internally encoded as an integer, stored effectively inside a database engine. Some RDBMS technologies actually support this kind of functionality [16]. Such mechanism is then totally transparent to end users, because columns remain visible with their original data types.

3.4 Adaptation of Existing Solutions

We use experiences of some finished or lasting projects and approaches to design a model meeting requirements formulated in Section 2, according to relatively standard modeling processes [18]. We adapted some solutions related to types of relations and multilingual sources from CERIF project [7]. We were also inspired by some entity-attribute-value (eav) variants [5] and some techniques of storing xml files in relational databases [4].

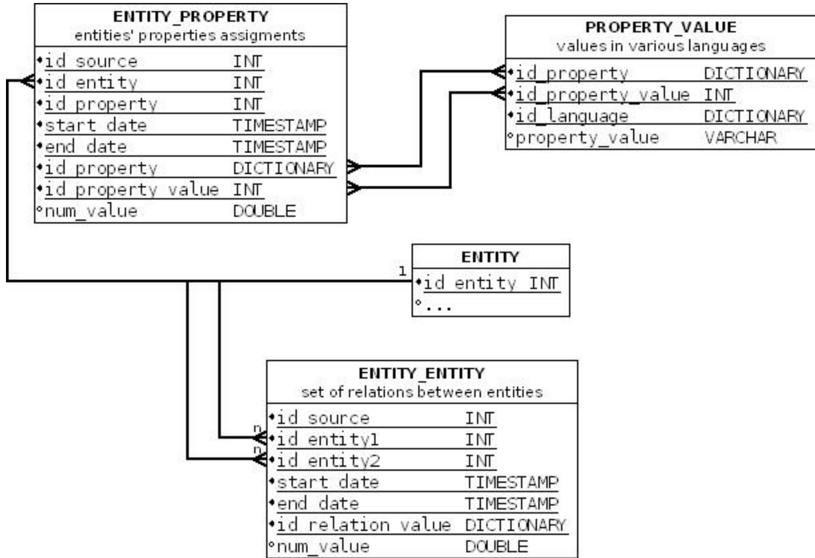

Fig. 1. Universal schema of assigning properties to entities and relations to entity pairs. In Section 4, we consider three sets of tables for three various entity semantics.

We store all properties of entities and all relations between entities in two tables. Thus, in order to add a new type of property or relation to the system, it is enough to insert a new row into one of tables displayed in Figure 1. This way, we avoid creating too many tables. We take the advantage of this universal solution in several areas of our model. The values of relations (e.g. 'IsAuthorOf', 'IsAffiliatedTo') are coded in DICTIONARY manner and stored in table `ENTITY_ENTITY`. The values of properties are coded as integers and stored in table `ENTITY_PROPERTY`. Their values can be decoded using table `PROPERTY_VALUE` with specific lingual context. For example, we can store the same person alias as 'Kolmogorov' (for English) and 'Kołmogorow' (Polish).

Where justified, there is temporality of described assignment for properties and relations taken into account. It is done by introducing validation time windows. There are two timestamps – `start_date` and `end_date`. If a given property or relation is not temporal, then we set up its corresponding `start_date` and `end_date` to minimum and maximum possible values, respectively.

There is always some tradeoff between effectiveness of storing data and easiness of their usage. One might consider our approach ineffective because of the size of tables `ENTITY_PROPERTY` and `ENTITY_ENTITY` but there are RDBMS solutions that can easily cope with this problem [16]. In future, in order to further increase effectiveness of a database engine, we can also consider some techniques of organizing data (such as partitioning, sorting, clustering), as well as some analytic algorithms that can work with faster approximate SQL queries.

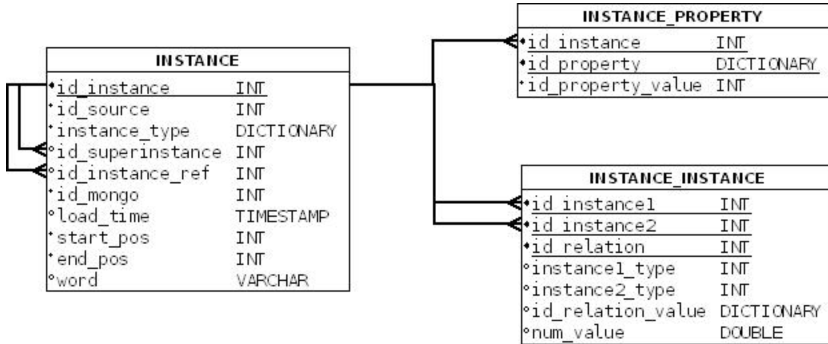

Fig. 2. Generic part of the proposed schema

4 Three Major Areas

4.1 Generic Area: *Instances*

This area contains only tables in which the primary keys include `id_instance`. Table `INSTANCE` corresponds to detailed information about *instances* – entities that we are able to distinguish while parsing the raw data (nxml files, which may result from various techniques of structural document segmentation and generally understood OCR [2,12]). At the current stage of our project these are: persons, documents, and organizations. This table contains information about structures of documents. In particular, we store information about documents decomposed onto parts such as abstract, section, bibliography et cetera. Each decomposed part is treated as a separate *instance* (together with information about its structural membership to a higher-level *instance*), which enables us to operate with the parts of documents when creating semantic indexes and conducting ad-hoc analytics. For example, investigation of trends in some domains of science may be enriched by the analysis of occurrences of *concepts* related to that domain in the concluding sections (summaries, directions for further research) of earlier articles followed by their analogous occurrences in technical sections of later articles. We go back to this topic in Section 4.4.

Some discussion how to efficiently store various xml-like structures in a relational database can be found, e.g., in [10]. The proposed schema refers to the state of the art in this area, although it contains also some specific solutions reflecting the nature and quality of the input data. Generally, we may expect two categories of *instances* – those corresponding to the (parts of) input articles (or input data related to other types of entities) and those extracted from the input articles but not corresponding directly to any of them. (Although they may later turn out to be *instances* of some *objects* already stored in the database; see Section 3.1.) For the sake of consistency and simplicity, we represent them in the same way, with some parts of metadata and structural information missing depending on category and quality of parsed information.

Table 1. Columns in table INSTANCE

column name	column type	note
id_instance	INT	id of (a part of) publication, person, institution
id_source	INT	it indicates which parsing method was applied
instance_type	DICTIONARY	type of *instance*, e.g.: document, person, abstract
id_superinstance	INT	id of the direct higher-level *instance* in hierarchy
id_instance_ref	INT	id of an *instance* that *instance* was parsed from
id_mongo	VARCHAR	id of the original document stored in MongoDB
load_time	TIMESTAMP	time of loading the package containing *instance*
start_pos	INT	the beginning of *instance's* range (see Figure 3)
end_pos	INT	the end of *instance's* range (see Figure 3)
word	VARCHAR	it is NULL for instance_type other than 'word'

Whenever some portion of text or information from other sources (e.g.: various forms of metadata) is classified as an *instance*, new records in table INSTANCE are generated, with new id_instance associated. As mentioned earlier, at this stage, we do not take into account that it may be a new *instance* of an *object* that already exists in the system. The attributes of table INSTANCE are displayed in Figure 2. In particular, we decided to use MongoDB [3] as the store of collected articles. Thus, column id_mongo refers to identifiers of the corresponding articles in the store. (It applies only to the first above category of *instances* – those corresponding to input articles.) However, we may think about this column in a more abstract way, as an identifier of original, not parsed content.

As already pointed out, table INSTANCE reflects the structure of publications and their decomposition onto parts (with words as the lowest hierarchy level). Hierarchy levels are encoded by column instance_type. Decomposition is performed by the parsing algorithms. It is reversible, i.e., no information is lost. Figure 3 illustrates relationship between an nxml file and its INSTANCE content.

When a new *instance* is detected during the parsing process, the piece of data which enables us to distinguish it is referred using column id_instance_ref. The details are available in Table 1. Thus, column id_instance_ref creates a link to *instances* of other *objects* detected in a given *instance* (e.g.: items in the bibliography). In particular, one may think about id_instance_ref as responsible for linkage between two above-mentioned categories of *instances* that are represented in a uniform way in our schema.

All metadata enclosed in input files or gained from other sources are stored in two tables: INSTANCE_PROPERTY and INSTANCE_INSTANCE. The latter one includes connections between persons and organizations, which should be revealed during analyzing publication (e.g.: affiliations), between organizations and publications (e.g.: editors of documents from references) and between persons and publications (e.g.: relation of being an author of publication).

In this part of the model, we also use the structure illustrated by Figure 1, for entity = *instance*.

```
<Document>
  <Abstract>
    This is an exemplary abstract.
  <\Abstract>
  <Section>
    In this section we present nothing.
  <\Section>
<\Document>
```

id_instance	id_superinstance	start_pos	end_pos	instance_type	word
1		0	13	Document	
2	1	0	6	Abstract	
3	2	0	1	word	This
4	2	1	2	word	is
5	2	2	3	word	an
6	2	3	4	word	exemplary
7	2	4	5	word	abstract
8	2	5	6	word	.
9	1	6	12	Section	
10	9	6	7	word	In
11	9	7	8	word	this
12	9	8	9	word	section
13	9	9	10	word	we
14	9	10	11	word	present
15	9	11	12	word	nothing
16	9	12	13	word	.

Fig. 3. An example of nxml file and its corresponding content in table `INSTANCE`

As already noted in Section 3.4, one might claim that it is unrealistic to put such a huge volume of data into a relational database schema. In order to verify it, we conducted a simple experiment with the RDBMS software introduced in [16], optimized with respect to data compression [9] and SQL-based analytics [14]. We parsed and loaded 5,000 scientific articles according to the proposed schema. The size of nxml representations of those articles was about 350 MB. We measured compression ratios for data stored in the applied RDBMS solution. Physical size of table `INSTANCE` turned out to be almost 40 times smaller than the corresponding tabular data obtained from the parsing algorithm. The sizes of tables `INSTANCE_INSTANCE`, `INSTANCE_PROPERTY`, and `PROPERTY_VALUE` were, respectively, 30, 20, and 5.5 times smaller than their corresponding inputs. We also measured performance of some examples of SQL statements reflecting, e.g., matching of *instances* of the same *objects*, or labeling *objects* with their most strongly represented *concepts*. Although 5,000 documents is significantly less than we should expect eventually in practice, we could clearly see that the observed SQL speed would be fully satisfactory even for far larger data.

4.2 Analytic Area: *Objects*

In this area we store tables, which are results of analytic algorithms performed on raw data stored in generic area. Again, we refer to the structure presented in Figure 1, this time for entity = *object*.

Let us start with table `OBJECT_MATCH`, which contains results of the matching algorithms, which group parts of *instances* (the whole *instance* is also treated as a 'part') in classes named *objects* and assign them with integer identifiers `id_object`. Thus, `OBJECT_MATCH` has two attributes: `id_instance`, which is `INSTANCE`'s primary key and `id_object`, which yields that every *instance's* part

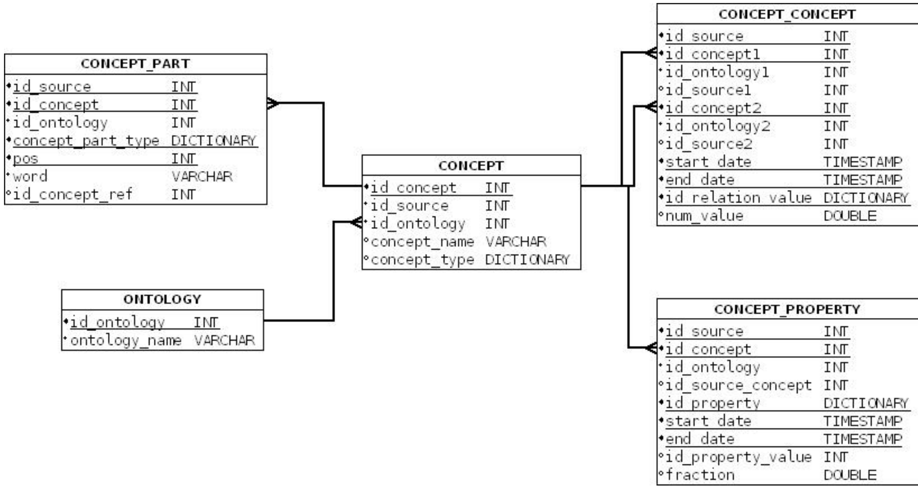

Fig. 4. Ontological part of the proposed schema

belongs to a group corresponding to an *object*. In the experimental phase of our project, we test multiple algorithms matching *instances*. We assume that `id_object` encodes a type of algorithm used to create the corresponding *object*. Eventually, once we verify which strategy is optimal, we will use a unique matching algorithm for the whole framework.

After all, analytic part of our relational database model is quite analogous to generic part displayed in Figure 2. The difference is that now we work at the level of `id_object` instead of `id_instance`. The corresponding attributes in tables describing *objects* are filled in by algorithms processing data in tables `OBJECT_MATCH` and `INSTANCE`. For example, table `OBJECT_OBJECT` stores information about relations between *objects* computed based on relations between *instances* in table `INSTANCE_INSTANCE`. This level is also a place for more intelligent algorithms that find a wider range of types and degrees of relationships between *objects* than it is present at the level of *instances*.

4.3 Ontological Area: *Concepts*

From syntactic point of view, one can treat ontologies as models of knowledge in which there can be distinguished entities, their properties and relations between them. This fits to our approach, therefore, we are going to store information retrieved from ontologies in tables similar to these described in Section 3.4. In case of ontologies, we will talk about *concepts* instead of *instances* (see Figure 4). Our major challenge in this area was to provide fairly universal framework for storing information about *concepts* acquired from different sources, such as, e.g., Wikipedia (full articles) or Wordnet (only lexems). Generally speaking, as a *concept* we treat every entity that can be retrieved from a source provided

CONCEPT_OBJECT relations between concepts and objects	
•id_source	INT
•id_object	INT
•id_concept	INT
°id_source_concept	INT
°id_ontology	INT
•start_date	TIMESTAMP
•end_date	TIMESTAMP
•id_relation_value	DICTIONARY
°num_value	DOUBLE

Fig. 5. Linking *concepts* and *objects* in table CONCEPT_OBJECT

by an expert (and using an algorithms accepted by an expert), which can have properties and some describable relations to other *concepts*.

Properties of *concepts* (e.g.: name of a *concept*) and relations between *concepts* are assumed to be temporal.

Apart from tables described in Section 3.4, we also need representation of *concepts* as texts. Such information is contained in table CONCEPT built similarly to INSTANCE. As one can represent a *concept* by different text descriptions, we introduce column concept_part_type. There is, however, a slight difference between this column and instance_type in table INSTANCE – as for now, we did not find a reason for introducing hierarchy for *concept* descriptions like in case of *instances*. Hence, concept_part_type is not the key in table CONCEPT. Also, we do not need to consider column concept_superpart.

We decided to separate the above subset of tables from generic part as we assume that ontologies will be loaded in more supervised way, so there is no place for noise or incompleteness. There is also a principal difference in usage of both parts. *Instances* reflect lower level information – we use them mainly for producing *objects*, not for retrieving new knowledge like in case of *concepts* and *objects* in the next subsection. Differences in data supply process are also the reason for distinguishing between *object* and *concept* areas.

4.4 Linking *Concepts* and *Objects*

Table CONCEPT_OBJECT (Figure 5) consists of relations between ontological *concepts* and analytic *objects*. One of the most important relations which we can derive, store and use is labeling documents or scientists with topics from ontologies (e.g. from Wikipedia). Going further, we can do it not only for the whole articles but also for their particular parts, which can lead to some interesting article structure-aware analytics. Querying the parts of documents is quite well-established area of research (see e.g. [11]). We have already mentioned about it in Section 4.1, although it may now make more sense for *objects* instead of their *instances*. One more example may be to reason about the most promising areas of science, e.g., by means a query formulated by a student who searches for potential topics of future thesis, where SQL-based heuristics may extract topics

occurring frequently in the concluding parts of articles but with no significant representation in bibliographies and major parts of articles.

Table `CONCEPT_OBJECT` is built analogously to `ENTITY_ENTITY` in Figure 1, where the first entity becomes analytic *object* and the second entity becomes ontological *concept*. The parts of *objects* (having the status of *objects* as well) can be also, e.g., equations, tables or figures in a publication. Thus, table `CONCEPT_OBJECT` may include a number of interesting relations, e.g., labeling specific figures from specified books with *concepts* such as 'Pythagorean theorem', or reflecting our previously discussed examples related to analyzing scientific trends and search for interesting scientific topics.

5 Conclusion

The presented model of storing and processing information related to scientific content has resulted from a number of design iterations that took into account various requirements, such as diversity and incompleteness of data sources, occurrence of multiple *instances* of the same *objects*, a need of dealing with potentially growing amount of types of *objects* without making the database schema overcomplicated, as well as analyzing *objects* with respect to various types relationships with *concepts* representing domain knowledge provided by experts or extracted semi-automatically from available sources. The obtained relational database schema contains core tables for three main layers – generic (*instances*), analytic (*objects*), and ontological (*concepts*) – in analogous formats, with ability to create additional intermediate tables and views whenever necessary.

Our major motivation to apply RDBMS framework was to provide a dual form of storing scientific content. We noticed that different tasks in our project required different characteristics of data access and processing, including massive comparisons of large collections of parts of documents. Ability to use SQL over clearly defined schema may open a variety of analytic possibilities. Appropriately chosen database technologies may enable to store huge amounts of parsed data and run SQL-based analytic tasks satisfactorily fast.

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

References

1. Agrawal, R., Ailamaki, A., Bernstein, P.A., Brewer, E.A., Carey, M.J., Chaudhuri, S., Doan, A., Florescu, D., Franklin, M.J., Garcia-Molina, H., Gehrke, J., Gruenwald, L., Haas, L.M., Halevy, A.Y., Hellerstein, J.M., Ioannidis, Y.E., Korth, H.F., Kossmann, D., Madden, S., Magoulas, R., Ooi, B.C., O'Reilly, T., Ramakrishnan, R., Sarawagi, S., Stonebraker, M., Szalay, A.S., Weikum, G.: The Claremont Report on Database Research. Commun. ACM 52(6), 56–65 (2009)

2. Betliński, P., Gora, P., Herba, K., Nguyen, T.T., Stawicki, S.: Semantic Recognition of Digital Documents. In: Bembenik, R., Skonieczny, Ł., Rybiński, H., Niezgódka, M. (eds.) Intelligent Tools for Building a Scientific Information Platform. Springer, Heidelberg (2011)
3. Chodorow, K., Dirolf, M.: MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. O'Reilly Media (2010)
4. Grust, T.: Accelerating XPath Location Steps. In: Proc. of Int. Conf. on Management of Data (SIGMOD), pp. 109–120 (2002)
5. Hammond, W., Stead, W., Straube, M.: A Chartless Record-Is It Adequate? In: Proceedings of the Annual Symposium on Computer Application in Medical Care, vol. 7, pp. 89–94 (1982)
6. Hellerstein, J.M., Stonebraker, M., Hamilton, J.R.: Architecture of a Database System. Foundations and Trends in Databases 1(2), 141–259 (2007)
7. Jörg, B., Jeffery, K., van Grootel, G., Asserson, A., Dvorak, J., Rasmussen, H.: CERIF, - 1.2 Full Data Model (FDM) Introduction and Specification (2008), http://www.eurocris.org/Uploads/Web/%20pages/CERIF2008/Release_1.2/CERIF2008_1.2_FDM.pdf
8. Kobdani, H., Schütze, H., Burkovski, A., Kessler, W., Heidemann, G.: Relational Feature Engineering of Natural Language Processing. In: Proc. of Int. Conf. on Information and Knowledge Management (CIKM), pp. 1705–1708 (2010)
9. Kowalski, M., Ślęzak, D., Toppin, G., Wojna, A.: Injecting Domain Knowledge into RDBMS – Compression of Alphanumeric Data Attributes. In: Kryszkiewicz, M., Rybinski, H., Skowron, A., Raś, Z.W. (eds.) ISMIS 2011. LNCS, vol. 6804, pp. 386–395. Springer, Heidelberg (2011)
10. Mihajlović, V., Blok, H.E., Hiemstra, D., Apers, P.M.G.: Score Region Algebra: Building a Transparent XML-R Database. In: Proc. of Int. Conf. on Information and Knowledge Management (CIKM), pp. 12–19 (2005)
11. Navarro, G., Baeza-Yates, R.A.: Proximal Nodes: A Model to Query Document Databases by Content and Structure. ACM Trans. Inf. Syst. 15(4), 400–435 (1997)
12. Nguyen, H.S., Ślęzak, D., Skowron, A., Bazan, J.G.: Semantic Search and Analytics over Large Repository of Scientific Articles. In: Bembenik, R., Skonieczny, Ł., Rybiński, H., Niezgódka, M. (eds.) Intelligent Tools for Building a Scientific Information Platform. Springer, Heidelberg (2011)
13. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A Comparison of Approaches to Large-scale Data Analysis. In: Proc. of Int. Conf. on Management of Data (SIGMOD), pp. 165–178 (2009)
14. Ślęzak, D., Eastwood, V.: Data Warehouse Technology by Infobright. In: Proc. of Int. Conf. on Management of Data (SIGMOD), pp. 841–846 (2009)
15. Ślęzak, D., Sosnowski, Ł.: SQL-Based Compound Object Comparators: A Case Study of Images Stored in ICE. In: Kim, T.-h., Kim, H.-K., Khan, M.K., Kiumi, A., Fang, W.-c., Ślęzak, D. (eds.) ASEA 2010. Communications in Computer and Information Science, vol. 117, pp. 303–316. Springer, Heidelberg (2010)
16. Ślęzak, D., Wróblewski, J., Eastwood, V., Synak, P.: Brighthouse: An Analytic Data Warehouse for Ad-hoc Queries. Proc. VLDB Endow. 1(2), 1337–1345 (2008)
17. Tekli, J., Chbeir, R., Yétongnon, K.: An Overview on XML Similarity: Background, Current Trends and Future Directions. Computer Science Review 3(3), 151–173 (2009)
18. Teorey, T., Lightstone, S., Nadeau, T.: Database Modeling & Design: Logical Design, 4th edn. Morgan Kaufmann (2005)