A Knowledge Representation and Data Provenance Model to Self-Tuning Database Systems

Ana Carolina Almeida - Sérgio Lifschitz - Karin Breitman
Departamento de Informática
PUC-Rio
Rio de Janeiro, Brazil
{abrito – sergio - karin} @inf.puc-rio.br

***Abstract*— Most autonomic database systems do not explicit their decision rationale behind tuning activities. Consequently, users may not trust some of the automatic tuning decisions. In this paper we propose a rather transparent strategy, that provides feedback to database administrators, based on information extracted from the database log. The proposed approach consists in transforming log results into a user-friendly knowledge representation, based on the graphical representation for OWL. This model provides users with the rationale behind system decisions, adds semantics to the database self-tuning actions, and provides useful provenance information about the whole process.**

Keywords- ontologies, transparency, self-tuning database.

I. INTRODUCTION

The task of self-tuning database systems involve automating the activities commonly done by database administrators (DBAs) to speed up database systems and application processing. Most of these systems keep the decision rationale behind parameter changes hidden from DBAs and end users. It is often the cast that self tuning systems are not adopted because DBAs do trust the tuning component decisions.

There is a growing interest in developing systems that provide transparency. Software transparency includes characteristics about information such as completeness, friendly, accessibility, objectivity, reliability, accuracy and consistency [12].

In this paper we propose a strategy that tackles the transparency issue. By providing user feedback, based on information extracted from the database logs, we provide a simple and intelligible way to represent tuning decisions.

The proposed approach consists in transforming log results into friendlier knowledge representation, based on the graphical representation for the OWL ontology language. This model provides users with better explanations about the system decisions, adds semantics to self-tuning actions, and provides useful provenance information about tuning process.

This paper is organized as follows: Section 2 comments on related works, Section 3 relates some important concepts, particularly about rationale and provenance; Section 3 briefly describes the self-tuning system to which this approach was applied, while Section 4 reports all steps involved in Knowledge Representation Model development. Finally, Section 5 concludes this paper with our final remarks and the future work.

II. RELATED WORK

While there is seminal work regarding rationale capturing and provenance in software engineering, this is not true regarding self-tuning databases [3, 17]. Indeed, we were no able to find literature pointing to a direct relationship involving rationale capture, provenance data and tuning database systems.

Nevertheless, some commercial databases may include self-tuning components that allow relating rationale and database tuning. For example, Oracle Database 10g has a self-tuning component called the SQL Tuning Advisor [18]. It receives one or more SQL statements as input and provides advice on how to optimize their execution plans. Furthermore, it gives the rationale for the advice, the estimated performance benefit and the actual command to implement the recommended advice. It relates to a collection of statistics on objects, creation of new indexes, restructuring the SQL statements, or even the creation of a SQL Profile. A user can choose if he or she accepts the recommendation to complete the tuning of the SQL statements.

There are also some proposals that relate provenance with databases. For example, the authors in [2] describe an approach to track the user's action while browsing database sources. Data are then copied into a curated database, in this case, applied to the bioinformatics context. The work in [8] brings a presentation data model with a higher level of abstraction that is located on top of the database logical schema in order to enhance usability. This high-level schema comprises only a small number of concepts. This model allows to the user to query the new schema summary directly. The authors in [8] also stress the importance of provenance and consistency across presentation models.

The Oracle SQL Tuning Advisor component, shows the rationale and detailed log about decisions but to a limited extent. It deals with automatic index dropping or re-creating indexes. Also, this component does not provide an actual knowledge model, that is, rationale with semantics. Neither does it provide rationale behavior in spite of having the provenance traceability.

Despite our efforts, we were not able to find literature that contemplated data provenance, transparency and rationale in respect to self-tuning database systems.

In what follows, we discuss some concepts that are relevant to understanding the importance of obtaining the design rationale behavior and the data provenance in the context of this paper.

III. DESIGN RATIONALE AND DATA PROVENANCE

Rationale methods aim at capturing, representing, and maintaining records about why developers have made the decisions they have, including the options they investigated, the criteria they selected to evaluate options, and, most important, the debate that lead to making decisions. Rationale can serve two different purposes: discourse and knowledge capture [1].

Design rationale includes background knowledge e.g., deliberating, reasoning, trade-off and decision-making in the design process of an artifact [5].

Design rationale capture is a technique that aims at providing a full description of decision making processes [3] by registering what decisions are made, when and why.

The rationale design approach can be applied to database design to obtain the explicit background knowledge that is usually only implicit in self-tuning database processes. It is useful because not only decision rationale is captured, but also dependencies and the justification behind the decisions' component system.

In fact, a great part of process knowledge is already captured by the tuning component and registered in the logs. In this paper we propose to use the logs to extract, organize and store rationale knowledge so as to provide an intelligible, user friendly, explanation of self-tuning actions.

In addition we are explore the possibilities of providing data provenance, that can be useful to the database administrators. Data provenance may be defined as "the source or origin of an object; its history and pedigree; a record of the ultimate derivation and passage of an item through its various owners." by The Oxford English Dictionary. In scientific experiments, provenance helps us interpret and understand results: by examining the sequence of steps that led to a result [7].

In respect to self-tuning databases, provenance information will helps us interpret and understand decision-making processes behind the design rationale behavior of the tuning component such as index creation, dropping or re-creation (reindex) and indexes decisions. With provenance information, database administrators will be able to identify weather the tuning component is performing according to the desired behavior.

The combination of design rationale and provenance information can be useful in helping database administrators understand the automatic decisions performed by the self tuning component and in the anticipating decisions, if necessary.

In the next section, we discuss self-tuning systems in more detail.

IV. SELF-TUNING SYSTEMS

One of the most important tasks of DBAs is to guarantee optimal response times to statements submitted by users of a very large DBMS (Database Management System). In [10] we have proposed a self-tuning tool to a relational DBMS that allows creating, dropping and recreating indexes automatically, in order to decrease SQL requests response times. Our tool extends existing proposals (e.g. [17] [11]) introducing automatic reindex the index structures through the investigation of the fragmentation level of indexes. We have implemented our architecture and ideas within the PostgreSQL RDBMS.

Our system's decisions are based on a set of heuristics that work on the expected benefits [9] [4] of a given index. Whenever the index benefit gets a negative value or a value that cannot justify its existence, the index could be dropped, or else it would only harm system's maintenance. When a workload is submitted to our self-tuning tool, all system component decisions are stored in a working file based on user submitted statements, as shown in Figure 1. For a given query we store their accumulated benefit and their creation cost. Our heuristics compare these 2 values, among others, to decide if the candidate (hypothetical) index should be automatically created or not.

This additional file generated, however, is not as friendly as a DBA could expect. Though, it has important information about the rationale behind the self-tuning component behavior.

It is usually not clear to most DBAs what have motivated the decisions behind automatic index management, especially if presented as a simple text file. Important information contained in these text files may include: candidate indexes, submitted statements and accumulated benefits, among others.

```
IndexAgent::POB::printDebug - Candidate Indexes:
  Name: hi_venda_0 | Table: venda | Accumulated Benefit: 8143.99
  | Times Used: 1 | Creation Cost: 57890.9
IndexAgent::POB::printDebug - Zero Created Indexes.
Query text: select prodnum, valor, data, qtd from venda where num = 10000;
```

Figure 1. Text File Example

```
IndexAgent::POB::printDebug - Candidate Indexes:
  Name: hi_venda_0 | Table: venda | Accumulated Benefit: 8143.99
  | Times Used: 1 | Creation Cost: 57890.9
IndexAgent::POB::printDebug - Zero Created Indexes.
Query text: select prodnum, valor, data, qtd from venda where num = 10000;
```

Figure 2. Text File Example

decisions earlier and faster. For example, the DBA would not have to wait for the automatic index creation if he could analyze the rationale and find out that the index was essential to the RDBMS submitted workload.

In the next Section we present the proposal of creating a knowledge representation model to provide more semantics and transparency about self-tuning actions based on logs.

V. KNOWLEDGE REPRESENTATION MODEL

In order to provide more semantics and transparency to the self-tuning tool actions, this paper proposes the use of ontology as a representation model in which to capture log information and redress it in a more user-friendly fashion. In what follows we describe the construction process for the ontology that captures the log information.

First, the terms used in text file generated by system and their concepts are identified and analyzed. From this analysis, the ontology is built. Figure 3. shows presents the central concepts in the ontology in a graphical way, while TABLE I. details theirs concepts.

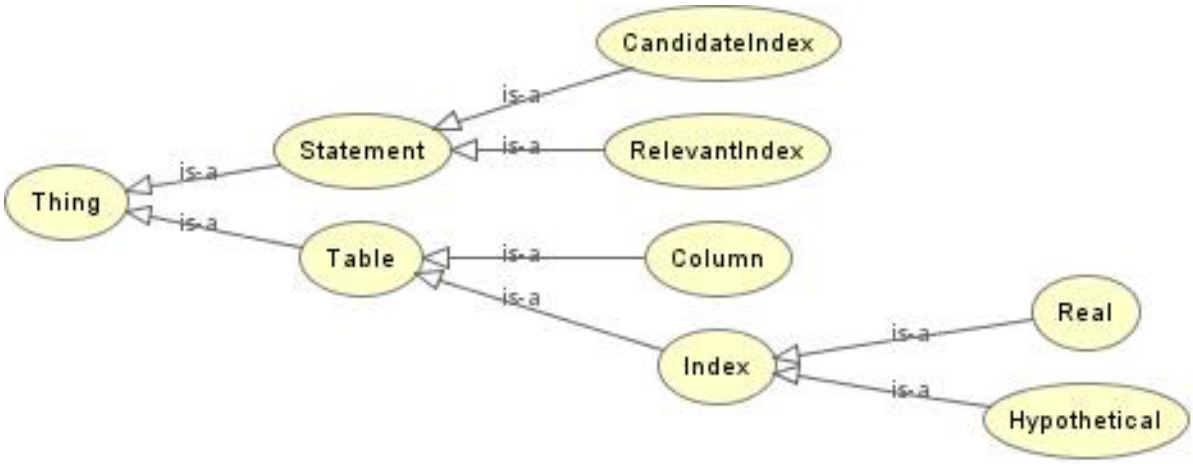

Figure 3. Self-tuning tool - Ontology model proposal

TABLE I. SELF-TUNING TOOL - DESCRIPTIONS OF TERMS

Term	Description
Statement	Statement may be a select (query), an insert, an update or a delete command in any structure (table, index, tuples) of database.
CandidateIndex	Hypothetical indexes for all relevant columns that can be used in the statement.
RelevantIndex	Real indexes for all relevant columns that can be used in the statement.
Table	A set of data elements (values).
Column	A set of data values of a particular simple type.
Index	A data structure that improves the speed of operations on a database table.
Real	Real index that has a physical materialization in the database.
Hypothetical	Hypothetical index that allows simulating the presence of an index in the database without its physical materialization.

Following, disjoint classes are created: Real and Hypothetical index, CandidateIndex and RelevantIndex. These classes are disjoint because an index can not be real and hypothetical at the same time. Also, object and data properties were elaborated and applied to classes and data (see TABLE II.).

TABLE II. OBJECT AND DATA PROPERTIES

Object Properties	
Property	***Description/Use example***
about	Used to show the relationship between index and column, that is "Index about Column". All indexes are created about one or many columns of table.
couldBe	Used to show the relationship between index and its type. For example: "Index couldBe Hypothetical" and "Index coulBe Real".
has	Used to show the relationship between statement and indexes or table and index or column. A statement can have indexes that are relevant or candidate to accelerate its execution. So, we created relationships as: "Statement has RelevantIndex" and "Statement has CandidateIndex". Also, a table can have indexes to accelerate queries about it (Table has Index) and columns as part of its composition (Table has Column).
is	Used to show the relationship between two objects that are similar because of their definitions. For example, all candidate indexes are hypothetical indexes created to evaluate their benefits about each query (CandidateIndex is Hypothetical). These candidate indexes are suggested by heuristic. Also, all relevant indexes are real indexes created because they are relevant to the statement (RelevantIndex is Real).
Data Properties	
Property	***Description***
aboutColumn	Columns that are used by relevant indexes. All relevant indexes are created about one or more columns in table.
aboutTable	Columns and indexes (candidate or real) are created about table.
accumulatedBenefit	Candidate and Relevant indexes have accumulated benefit that is used by component system to decide to create or to drop this index or not.
atPosition	Columns and relevant indexes are created at the physically position on the table.
Bonus	Candidate and Relevant indexes have bonus that is used by component system to show the index bonus in the statement execution.
Cost	The execution statement cost.
creationCost	Candidate index has creation cost that is the cost of the creation of this index when it turns real.
dropSituation	How many times the relevant indexes are dropped.
eliminationCost	Relevant indexes elimination cost to the database.
executionNumber	The statement execution number. The statement can be executed one or more times in the same workload.
firstUsageNotification	Notification about first usage of a relevant index.
Pages	How many pages the table uses physically.
rowsProcessed	How many table rows need to be processed to execute the statement.
Scans	How many table scans need to be did to execute the statement.
Statement	The statement that is submitted against the database.
timesUsed	How many times the candidate index is used by the database to execute a statement.
tuples	How many tuples the table has.

The object properties are used to relate classes to other classes in the ontology. They are used to explicit the relationship between indexes and columns as well as to statements in the ontology.

The data properties, on the other hand, are attributes of the classes themselves. They are very useful to provide additional information about the columns, e.g., the ones that are used by such and such indexes. Attributes play a central role in capturing information used to mine the self-tuning mechanism rationale, as they provide more details of the self tuning mechanism behavior.

Once the ontology is defined, we have developed a script to create RDF triples. The log of the system transformed to the RDF notation to facilitate processing by the semantic system, designed to extract the design rationale and provenance. The system is detailed in the following section.

VI. SEMANTIC SYSTEM

We have developed a *Perl* script to map the log and create RDF triples. These triples provide more semantics about the database log due to the ontology used. It should be noted that other self-tuning systems can use the same ontology. Similarly, other systems can use the semantic system implemented.

In our case, the RDF triples created may be stored in the virtuoso database [15] [13] that fully supports sparql queries. We are currently developing a semantic system using python language [14] that access the virtuoso database and shows the information about the log system in friendlier manner. Figure 3 illustrates the information related to the hi_new_order_1 *created index*.

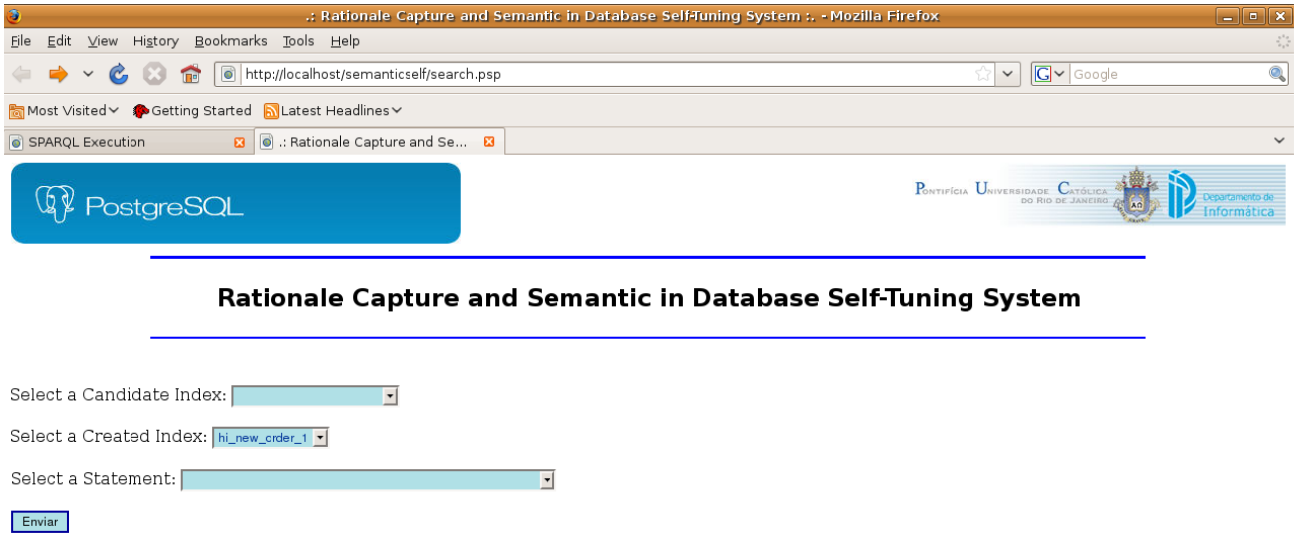

Figure 4. Semantic system main page – Created Index Query

In Figure 4 we illustrate how the semantic system shows the rationale about a given index. In this particular case the system shows the accumulated benefit, creation cost, execution number and the statement used by this index.

Figure 5 shows that the index, first as hypothetical, has an accumulated benefit of 158.6. During the execution of other SQL statements we can analyze that the accumulated benefit is ranging according to the specific statement that is submitted to the database. Note that, when a highly accumulated benefit is reached (e.g., last execution with 780.518 accumulated benefit value) the self-tuning component decides to create this index, as it would probably reduce query execution costs.

One should not misunderstand “execution” with the “execution number”. The first one is the execution number each time the component system shows the index as relevant to the statement. On the other hand, the “execution number” label (or tag) is the statement number that is submitted to the database.

There is a difference in the last block between the “execution” and the “execution number” information. We explain this with the presence of a statement submitted to the database that the self-tuning component did not consider relevant in respect to the index.

In the future, more database information can be added to the log system to provide additional information to users. Indeed, we are extending the semantic system to provide a graphic to show the accumulated benefit evolution.

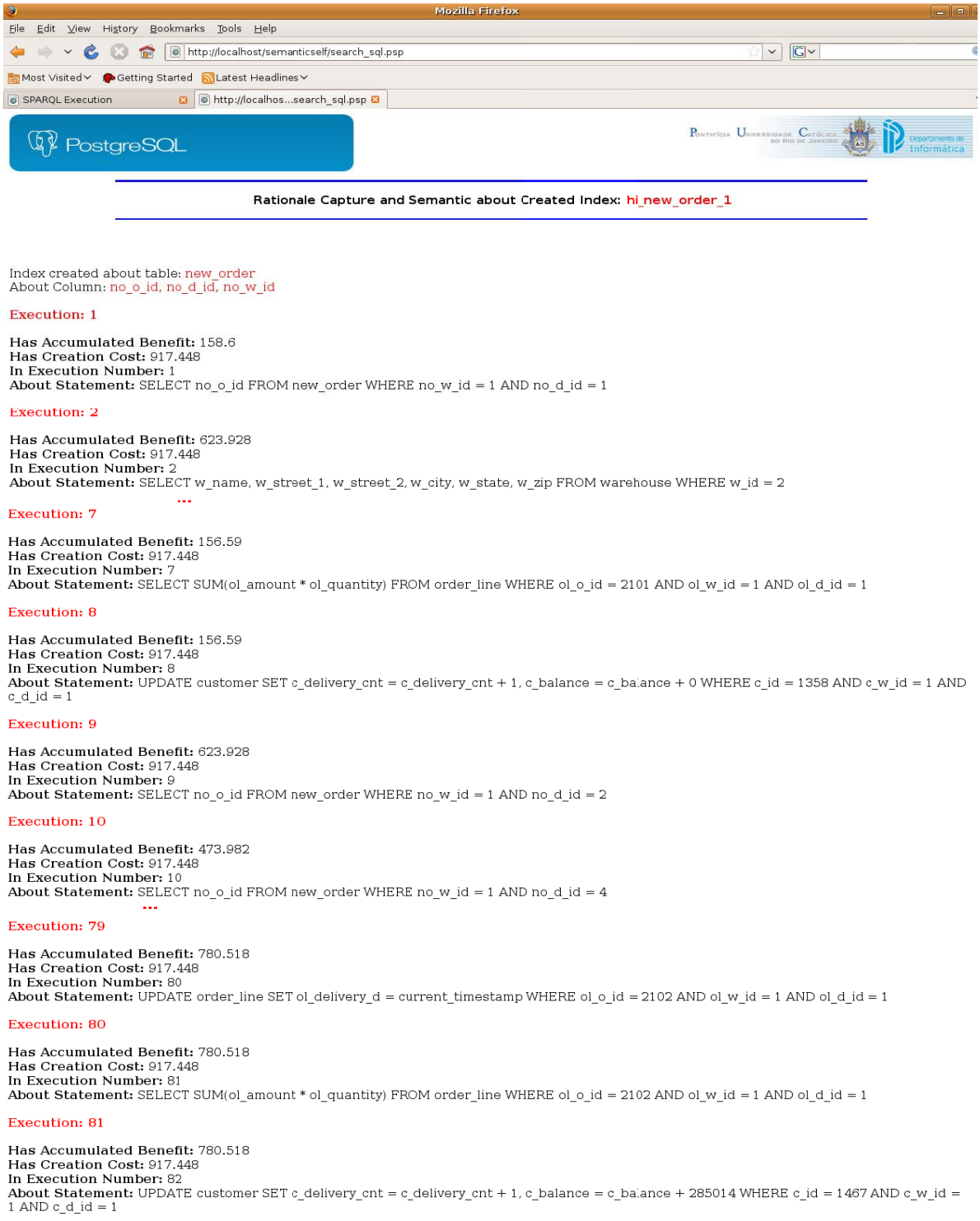

Figure 5. Semantic system – Real Index Query Results

VII. CONCLUSIONS

In this paper the problem of transparency behind decisions made by database self-tuning tools is addressed. Most commercial systems make important decisions without providing users the rationale that supports their reasoning.

The contribution of this paper is providing tool strategy that tackles the transparency issue. By providing user feedback, based on information extracted from the database logs, we provide a simple and intelligible way to represent tuning decisions.

The proposed approach maps the self-tuning tool execution log to an ontology and creates RDF triples that provide semantic concepts that both describe and backup the system self-tuning component decisions. Particularly useful is to provide users with provenance data and reasoning to explain index automatic creation or dropping.

The proposed approach can be extended to cover other self-tuning applications to provide semantic concepts as [[6]]. Also, this approach can be improved to show, either periodically or by demand, reasoning behind other automatic decisions, given different scenarios and workloads. For instance, a DBA could anticipate the creation of an index before the self-tuning component would decide upon doing so.

REFERENCES

[1] J. Bosch, "Software architecture: the next step", in: Proceedings 1st European Workshop on Software Architecture (EWSA'04), Springer-Verlag, pp. 194—199, 2004.

[2] P. Buneman, A. P. Chapman, J. Cheney, "Provenance management in curated databases", in: SIGMOD 2006, pp. 539-550, Chicago, Illinois, USA, 2006.

[3] E. J. Conklin, K. C. B. Yakemovic, "A process-oriented approach to design rationale", in: Human-Comput. Interaction, 6(3–4), pp. 357—391, 1991.

[4] R. L. C. Costa, S. Lifschitz, M. V. Salles, "Index self-tuning and agent-based databases", in: Proceedings of the Latin-American Conference on Informatics (CLEI), p. 76, Abstracts Proceedings; 12 pp. CD–ROM Proceedings, 2002.

[5] A. H. Dutoit, B. Paech, "Rationale management in software engineering", in: Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing Company, p. 92, 2000.

[6] J. M. S. M. Filho, "A non-intrusive approach to automatic maintenance of database physical design" (in portuguese), Ph.D.'s Thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2008.

[7] J. Freire, D. Koop, E. Santos, C. T. Silva, "Provenance for computational tasks: a survey", in: IEEE Computing in Science & Engineering, 10(3), pp. 11—21, 2008.

[8] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, C. Yu, "Making database systems usable", in: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 13-24, 2007.

[9] G. Lohman, G. Valentin, D. Zilio, M. Zuliani, A. Skelley, "DB2 advisor: an optimizer smart enough to recommend its own indexes", in: Proceedings of the IEEE International Conference on Data Engineering (ICDE), pp. 101—110, 2000.

[10] E. Morelli, J. M. Monteiro, A. C. Almeida, S. Lifschitz, "Automatic Reindexing in Relational DBMS" (in portuguese), in: XXIV Database Brazilian Symposium (SBBD), pp. 31-45, Fortaleza, 2009.

[11] Costa, R. L. C., Lifschitz, S., Noronha, M. and Salles, M. V., "Implementation of an Agent Architecture for Automated Index Tuning", in Proceedings of the ICDE Workshops, p. 1215, 2005.

[12] A. P. Oliveira, C. Cappelli, H. S. Cunha, J. C. S. P. Leite, V. M. B. Werneck, "Engenharia de requisitos intencional: tornando o software mais transparente", in: SBES'07, http://www.sbbd-sbes2007.ufpb.br/tuto3.pdf, 2007.

[13] Openlink software - universal server plataform for the real-time enterprise, http://virtuoso.openlinksw.com/, 2009.

[14] Python software foundation: python programming language – official website". Available in: http://www.python.org/, 2009.

[15] J. Rapoza, "OpenLinks virtuoso has many talents" in: eWeek magazine, 2004.

[16] W. C. Regli, X. Hu, M. Atwood, W. Sun, "A survey of design rationale systems: approaches, representation, capture and retrieval", in: Engineering with Computers, vol. 16, pp. 209--235, Springer-Verlag London Limited, 2000.

[17] Salles, M. A. V., Morelli, E.T., Lifschitz, S, "Towards Autonomic Index Maintenance", in: XX Brazilian Symposium on Database (SBBD), pp. 176-190, 2006.

[18] S. Surapaneni, "Automatic SQL tuning using SQL tuning advisor", in: Database Journal, The Knowledge Center for Database Professionals, Chicago, 2005.