A Tool for Analyzing and Tuning Relational Database Applications: SQL Query Analyzer and Schema EnHancer (SQUASH) *

Andreas M. Boehm[†] Matthias Wetzka[‡] Albert Sickmann[§] Dietmar Seipel[¶]

Abstract

A common problem in using and running RDBMS is performance, which highly depends on the quality of the database schema design and the resulting structure of the tables and the logical relations between them. In production reality, the performance mainly depends on the data that is stored in an RDBMS and on the semantics comprised in the data and the corresponding queries.

We implemented a system based on SWI-Prolog. The system is capable of importing any relational database schema and any SQL statement from an XML representation. This information can be queried and transformed, thus allowing modification and dynamic processing of the schema data. Visualization of relationships and join paths is integrated, too. Using an online connection, SQUASH queries current data from the RDBMS, such as join selectivities or table sizes.

The system allows for tuning the database schema according to the load profile induced by the application. SQUASH proposes changes in the database schema such as the creation of indexes, partitioning, splitting or further normalization. SQL statements are adapted simultaneously upon modification of the schema. SQL statements are rewritten in consideration of RDBMS-specific rules for processing statements including the integration of optimizer hints. The resulting statements are executed more efficiently, thus reducing application response times.

1 Introduction

While in productive use, enterprise-class databases undergo a lot of changes in order to keep up with ever-changing requirements. The growing space-requirements and the growing complexity of a productive database increases the complexity of maintaining a good performance of the database query execution. Performance is highly dependant on the database schema design [12]. In addition, a complex database schema is more prone to design errors.

Increasing the performance and the manageability of a database usually implies restructuring the database schema and has effects on the application code. Additionally, tuning the query execution is associated with adding secondary data structures such as indexes or horizontal partitioning [2, 7, 15]. Because applications depend on the database schema, its modifi cation implies the adaption of the queries used in the application code. The tuning of a complex database is usually done by specially trained experts having good knowledge of database design and many years of experience tuning databases [18, 24]. In the process of optimization, many characteristic values for the database need to be calculated and assessed in order to determine an optimal confi guration. Special tuning tools can help the DBA to focus on the important information necessary for the optimization and can support the DBA in complex

*This work was supported by the Deutsche Forschungsgemeinschaft (FZT 82).

[†]Protein Mass Spectrometry and Functional Proteomics Group, Rudolf-Virchow-Center for Experimental Biomedicine, Universitaet Wuerzburg, Versbacher Strasse 9, D-97078 Wuerzburg, Germany

[‡]Department of Computer Science, University of Würzburg, Am Hubland, D-97074 Würzburg, Germany

[§]Protein Mass Spectrometry and Functional Proteomics Group, Rudolf-Virchow-Center for Experimental Biomedicine, Universitaet Wuerzburg, Versbacher Strasse 9, D-97078 Wuerzburg, Germany

[¶]Department of Computer Science, University of Würzburg, Am Hubland, D-97074 Würzburg, Germany

database schema manipulations [18, 24]. More sophisticated tools can even propose optimized database configurations by using the database schema and a characteristic workload log as input [1, 6, 13].

We have defined an XML representation of SQL schema definitions and SQL queries, allowing for processing by means of SWI-Prolog [9, 25]. An XML-based format named SquashML was chosen for the representation of SQL-data because of its extensibility, flexibility and hierarchical organization. All syntactic elements in SQL are expressed by XML-elements, yielding a hierarchical representation of the information. The core of SquashML is predetermined by the SQL standard, but SQL also allows for DBMS-specific constructs that yield different dialects of SQL. It is extensible, as for example some definitions and storage parameters from the Oracle™ DBMS were integrated. SquashML is able to represent schema objects, as well as database queries obtained from application code or from logging data. This format allows for easily processing and evaluation of the database schema information. Currently, supported schema objects include table and index definitions. In Prolog, the representation of XML documents can be realized by using the Field Notation (FN) as data structure [21]. FNQuery [21] is a Prolog-based query and manipulation language for XML documents represented in FN, which embeds many features of XQuery [5].

2 Database Schema Refactoring and Application Tuning

Based on the XML-based representation, algorithms have been developed providing functions for visualization, analysis, refactoring and tuning of database schemas and the queries of the corresponding application. A combination of SWI-Prolog [25] and Perl [8] was used for the implementation of the algorithms as well as for im- and exporting the SQL-data.

The Squash-Analyzer is divided into four components used for visualization, analysis, manipulation and optimization. Using the visualization and analysis components, a DBA can gain a quick overview of the most important characteritics of the database and use this information for manual tuning decisions. The manipulation component provides features for carrying out complex manipulations of the database schema. The key feature of this component consists of the automatic propagation of schema changes and applying them to the queries of the application. The optimization component accesses the data of the analysis component and uses heuristic algorithms to automatically determine an optimized configuration of the database schema. It uses the manipulation component of Squash in order to apply the suggested changes to the XML-representation of the database schema.

The Squash-Analyzer is part of the DisLog Developers' Toolkit (DDK). It has been implemented in SWI/XPCE-Prolog [26] and can be loaded into any application written in SWI/XPCE-Prolog on either Windows™, UNIX or Linux. The whole system is embedded into a framework that allows for im- and exporting of SQL-code of database schema as well as import of a workload log and makes the functions of the system accessible through a GUI.

2.1 Refactoring Database Schemas and Queries

The Squash system provides functions to view the different aspects of the database schema with visual markup of important properties. In addition, the system is able to analyze a database schema and a corresponding workload for semantic errors, flaws and inconsistencies.

Visualization. Comprehending the structure of a complex database schemas just by reading the SQL create statements is a very demanding task and design errors can be missed easily. Squash provides a number of different visualization methods for the database schema and queries. Complex select statements tend to include many tables and use them in join operations. Therefore, Squash uses a tree representation for query visualization. If a select statement consists of nested subqueries, then these queries can be included into the view.

Analysis. The analysis component of Squash performs multiple tasks. Firstly, it can be used by the DBA

to gain a quick overview of the most important aspects of the schema. In addition, Squash also checks the database schema and the queries for possible design flaws or semantic inconsistencies. Using this information, the DBA can manually decide on optimization methods for the database schema. Secondly, when using the automatic optimization methods of Squash, these data is used as input for the heuristic algorithms which are described later in this section. Additionally, the analysis component collects schema properties and displays them. These functions are used in order to create reports of the database schema and the queries.

The more advanced algorithms of the analysis component check for common design flaws and semantic errors in the database schema as well within the queries. Especially within queries, semantic errors are often introduced by inexperienced database programmers [4, 14]. In Squash, methods for detecting common semantic errors in queries are implemented. This includes the detection of constant output columns, redundant output columns, redundant joins, incomplete column references, and the detection of joins lacking at least one foreign-key relationship.

The database schema can also be checked for design flaws by Squash. The available functions include the detection of isolated schema parts, cyclic foreign key references, unused columns as well as unused tables, unique indexes that are defined as non-unique, evaluating the quality of existing indexes, datatype conflicts, anonymous primary- as well as foreign-key constraints, missing primary keys, and the calculation of the estimated best join order.

Refactoring and Manipulation. The refactoring component allows for manipulating the database schema and the corresponding application code. Besides trivial manipulations such as adding or removing colums, the system also supports complex schema manipulations that affect other parts of the database schema and the application queries, for example, vertical partitioning and merging of tables. Squash is able to provide all queries in the workload with the best index set according to the heuristic function of the system, and it generates appropriate hints.

2.2 Physical Database Optimization

The optimization component of Squash supports the proposal and the creation of indexes as well as horizontal partitions. The system analyzes the database schema in conjunction with the workload and generates an optimized configuration. Many problems in database optimization, such as determining an optimal set of indexes, are known to be NP-complete [10, 16, 20]. Since exact optimality is much too complex to compute, heuristic algorithms are used in order to determine a new configuration of the database schema which is expected to perform better. Squash uses a similar heuristic multi-step approach in order to find good configurations. In the first step, statistical data are collected from the database. This information is used to minimize the set of columns that contribute to an optimized configuration. In the second step, the workload is analyzed and the quality is calculated for each column. All columns whose quality is below a threshold are removed from the set of candidate columns. Finally, the remaining columns are evaluated in detail and a solution is generated and presented. These algorithms were developed without a special DBMS in mind, but they are parameterized for application to Oracle™.

Index Proposal. The index proposal algorithm is able to propose and generate multi-column indexes. The solution-space is reduced to a manageable size, by a partitioned multi-step approach in combination with a scoring function. However, the set of columns is still unordered. Therefore, the algorithm sorts the columns in an order being most useful for the index. Squash offers three different sorting methods. The first one orders the columns by decreasing selectivity. The second method sorts them according to the type of condition they are used in. The third method reverses the order in which the columns appear in the query. Which one produces the best results, depends on the query optimizer of the DBMS. In the case study, sorting according to the `WHERE` clause was found to yield good results with Oracle™.

Horizontal Partitioning. The core task for generating a good proposal for horizontal partitioning consists of determining a column of the table being partitioned, that is suitable for partitioning. In the case of range partitioning, the borders of each partition are calculated by analysing the histograms of the

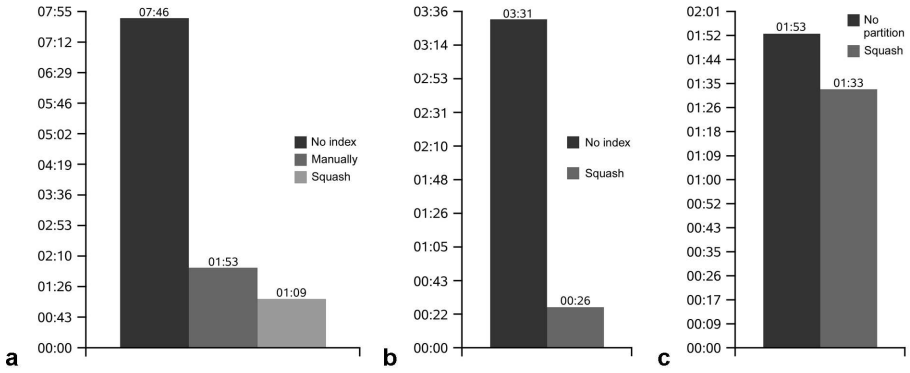

Figure 1: *Runtimes of using indexes and partitions proposed by Squash.*
a: *Sequest™ statement: The duration was 7:46 min without any indexes. This could be reduced by manual tuning to 1:53 min, whereas tuning by Squash achieved a further reduction of about 49% down to 1:09 min, 14% of the original execution time.*
b: *Mascot™ statement: The execution time was 3:31 without any indexes. This was not tuned manually before. Tuning by Squash and application of its suggested indexes yielded a reduction to 0:26 min.*
c: *Sequest™ statement: The execution time was 1:53 min (manually tuned), whereas tuning by Squash and application of partitions achieved further reduction of about 19% to 1:33 min.*

partition key. The number of partitions for hash partitioning is calculated from the table volume.

2.3 Case Study

A case study was conducted, using a database-based system designed for proteomics [27] based on mass spectrometry (MS). It is designed for storing data obtained from MS-based proteomics, providing support for large scale data evaluation [28]. This system, composed of the two subsystems `seqDB` and `resDB` [3, 28], was temporarily ported to Oracle™ for being analyzed by Squash. The complete database schema consists of 46 tables requiring about 68.5 GB disk space, and it fulfills the definition of a data ware house system. In addition, it supports data mining. We found interesting results for two characteristic statements of the data evaluation part `resDB`, that were obtained from an application log and were submitted to Squash for analysis in conjunction with the schema creation script. The first statement performs the grouping of peptide results into protein results that were obtained by Sequest™ [11], the second performs the same task for results of Mascot™ [19]. For demonstration, the Sequest™ statement was previously tuned manually by an experienced database administrator, the Mascot™ query was left completely untuned. The results of the tuning are depicted in Figure 1.

3 Conclusions

We presented a tool named Squash-Analyser that applies methods derived from artificial intelligence to relational database design and tuning. Input and ouput are XML-based, and operations on these data are performed using FNQuery and the DisLog Developers' Toolkit [22, 23].

Existing XML representations of databases like SQL/XML usually focus on the representation of the database contents, i.e. the tables, and not on the schema definition itself [17]. SquashML was developed specifically to map only the database schema and queries, without the current contents of the database. It is a direct mapping from SQL-`SELECT` and -`CREATE` statements into XML. This allows for an easy implementation of parsers that transform SQL code from various dialects into SquashML and vice versa.

The use of the Squash-Analyser allows for refactoring of database applications and considers interactions between application code and database schema definition. Both types of code can be handled and manipulated simultaneously. Thus, application maintenance is simplified and the chance for errors is reduced, in sum yielding time and resource savings.

References

[1] AGRAWAL, S. ; NARASAYYA, V. ; YANG, B.: Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. In: *International Conference on Management of Data*. Paris, France : ACM Press New York, NY, USA, 2004, pp. 359 – 370

[2] BELLATRECHE, L. ; KARLAPALEM, K. ; MOHANIA, M. K. ; SCHNEIDER, M.: What Can Partitioning Do for Your Data Warehouses and Data Marts? In: *IDEAS '00: Proceedings of the 2000 International Symposium on Database Engineering & Applications*. Washington, DC, USA : IEEE Computer Society, 2000, pp. 437–446

[3] BOEHM, A. M. ; SICKMANN, A.: A Comprehensive Dictionary of Protein Accession Codes for Complete Protein Accession Identifier Alias Resolving. In: *Proteomics* accepted (2006)

[4] BRASS, S. ; GOLDBERG, C.: Proving the Safety of SQL Queries. In: *5th International Conference on Quality of Software*, 2005

[5] CHAMBERLIN, D.: XQuery: a Query Language for XML. In: IVES, Z. (Ed.) ; PAPAKONSTANTINOU, Y. (Ed.) ; HALEVY, A. (Ed.): *International Conference on Management of Data (ACM SIGMOD)*. San Diego, California : ACM Press New York, 2003, pp. 682–682

[6] CHAUDHURI, S. ; NARASAYYA, V.: Autoadmin What-If Index Analysis Utility. In: TIWARY, A. (Ed.) ; FRANKLIN, M. (Ed.): *International Conference on Management of Data archive. Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. Seattle, Washington : ACM Press, New York, NY, USA, 1998, pp. 367–378

[7] CHOENNI, S. ; BLANKEN, H. M. ; CHANG, T.: Index Selection in Relational Databases. In: ABOU-RABIA, O. (Ed.) ; CHANG, C. K. (Ed.) ; KOCZKODAJ, W. W. (Ed.): *ICCI '93: Proceedings of the Fifth International Conference on Computing and Information*, IEEE Computer Society, Washington, DC, USA, 1993, pp. 491 – 496

[8] CHRISTIANSEN, T.: *CGI Programming in Perl*. 1. Tom Christiansen Perl Consultancy, 1998

[9] CLOCKSIN, W. F. ; MELLISH, C. S.: *Programming in Prolog*. 5. Berlin : Springer, 2003

[10] COMER, D.: The Difficulty of Optimum Index Selection. In: *ACM Transactions on Database Systems* 3 (1978), Nr. 4, pp. 440–445

[11] ENG, J. K. ; MCCORMACK, A. L. ; YATES, J. R.: An Approach to Correlate Tandem Mass Spectral Data of Peptides with Amino Acid Sequences in a Protein Database. In: *Journal of the American Society for Mass Spectrometry* 5 (1994), Nr. 11, pp. 976–989

[12] FAGIN, R.: Normal Forms and Relational Database Operators. In: *Proceedings of the ACM-SIGMOD Conference* (1979)

[13] FINKELSTEIN, S. ; SCHKOLNICK, M. ; TIBERIO, P.: Physical Database Design for Relational Databases. In: *ACM Transactions on Database Systems (TODS)* 13 (1988), Nr. 1, pp. 91 – 128

[14] GOLDBERG, C. ; BRASS, S.: Semantic Errors in SQL Queries: A Quite Complete List. In: *Tagungsband zum 16. GI-Workshop Grundlagen von Datenbanken*, 2004, pp. 58–62

[15] GRUENWALD, L. ; EICH, M.: Selecting a Database Partitioning Technique. In: *Journal of Database Management* 4 (1993), Nr. 3, pp. 27–39

[16] IBARAKI, T. ; KAMEDA, T.: On the Optimal Nesting Order for Computing N-Relational Joins. In: *ACM Transactions Database Systems* 9 (1984), Nr. 3, pp. 482–502

[17] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO/IEC 9075-14:2003 Information Technology – Database Languages – SQL – Part 14: XML-Related Specifications (SQL/XML)*. International Organization for Standardization, 2003

[18] KWAN, E. ; LIGHTSTONE, S. ; SCHIEFER, B. ; STORM, A. ; WU, L.: Automatic Database Configuration for DB2 Universal Database: Compressing Years of Performance Expertise into Seconds of Execution. In: WEIKUM, G. (Ed.) ; SCHÖNING, H. (Ed.) ; RAHM, E. (Ed.): *10. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW, Datenbanksysteme für Business, Technologie und Web)* Bd. 26. Leipzig : Lecture Notes in Informatics (LNI), 2003, pp. 620–629

[19] PERKINS, D. N. ; PAPPIN, D. J. C. ; CREASY, D. M. ; COTTRELL, J. S.: Probability-Based Protein Identification by Searching Sequence Databases Using Mass Spectrometry Data. In: *Electrophoresis* 20 (1999), Nr. 18, pp. 3551–3567

[20] ROZEN, S. ; SHASHA, D.: A Framework for Automating Physical Database Design. In: *VLDB 1991: Proc. of the 17th International Conference on Very Large Data Bases*, Morgan Kaufmann, 1991, pp. 401–411

[21] SEIPEL, D.: *Processing XML-Documents in Prolog*. Proc. 17th Workshop on Logic Programmierung (WLP 2002), 2002

[22] SEIPEL, D. ; BAUMEISTER, J. ; HOPFNER, M.: Declarative Querying and Visualizing Knowledge Bases in XML. In: *15th International Conference of Declarative Programming and Knowledge Management (INAP 2004)*, 2004, pp. 140–151

[23] SEIPEL, D. ; PRÄTOR, K.: XML Transformations Based on Logic Programming. In: *18th Workshop on Logic Programming (WLP 2005)*, 2005, pp. 5–16

[24] TELFORD, R. ; HORMAN, R. ; LIGHTSTONE, S. ; MARKOV, N. ; O'CONNELL, S. ; LOHMAN, G.: Usability and Design Considerations for an Autonomic Relational Database Management System. In: *IBM Systems Journal* 42 (2003), Nr. 4, pp. 568–581

[25] WIELEMAKER, J.: An Overview of the SWI-Prolog Programming Environment. In: MESNARD, F. (Ed.) ; SEREBENIK, A. (Ed.): *13th International Workshop on Logic Programming Environments*. Heverlee, Belgium : Katholieke Universiteit Leuven, 2003, pp. 1–16

[26] WIELEMAKER, J.: *SWI-Prolog*. Version: 2005. http://www.swi-prolog.org/

[27] WILKINS, M. R. ; SANCHEZ, J.-C. ; GOOLEY, A. A. ; APPEL, R. D. ; HUMPHERY-SMITH, I. ; HOCHSTRASSER, D. F. ; WILLIAMS, K. L.: Progress with Proteome Projects: Why All Proteins Expressed by a Genome Should Be Identified and How to Do It. In: *Biotechnology and Genetic Engineering Reviews* (1996), Nr. 13, pp. 19–50

[28] ZAHEDI, R. P. ; SICKMANN, A. ; BOEHM, A. M. ; WINKLER, C. ; ZUFALL, N. ; SCHÖNFISCH, B. ; GUIARD, B. ; PFANNER, N. ; MEISINGER, C.: Proteomic Analysis of the Yeast Mitochondrial Outer Membrane Reveals Accumulation of a Subclass of Preproteins. In: *Molecular Biology of the Cell* 17 (2006), Nr. 3, pp. 1436–1450