SOCIETY FOR SCIENCE AND EDUCATION UNITED KINGDOM
TMLAI TRANSACTIONS ON MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE
VOLUME 2, ISSUE 5
ISSN: 2054 - 7390

Proxy Framework for Database Extension: Database-Independent Function Extension

Pinaet Phoonsasrakun[1], Alexander Adam[2], Attaporn Khaesawad[1] and Seree Chinodom[1]
[1]Department of Information Technology, Burapha University Sakaeo Campus, 27160 Sakaeo, Thailand;
[2]Development Division, dimensio Informatics GmbH, 09111 Chemnitz, Germany
pinaet@buu.ac.th; alad@dimensio-informatics.com; attaporn@buu.ac.th; seree@buu.ac.th

ABSTRACT

This paper presents a proxy framework, which allows introducing various functionalities in an existing database system and application environment. By using a proxy not interfering with the application and with the database are necessary. An abstraction layer is introduced to make functional modules, database independent, and illustrates its use in two examples – index integration and a result cache. Finally, an overview of the results that have been achieved is given.

Keywords: Database Extension, Indexing, Proxy, Transparent.

1 Introduction

Database systems are designed to support a large number of applications. Where this support is not given, they were extended. For example, SQL continuously developed and the manufacturers add even non-standard extensions to it. Some examples of this are the geodetic extensions of IBM DB2 [1] and Oracle [2]. In addition to these built-in extensions that are available for the total number available, manufacturers allow the user-specific expansion of their systems. This includes, for a long time, user-defined data types and functions [3, 4]. Applications, these possibilities do not go far enough that even does data storage, i.e. how the data is to be stored and how the indexing has to take place on them, have the right to decide [5, 6].

The possibilities enumerated here are used partially by different groups of database users. On the one hand, application vendor draw their benefits from user-defined types and functions. On the other hand with skilful distribution of data, indexes, and other parameters, database administrators try to get the most out of the individual systems.

Besides, both sides can run into problems. For example, an application developer who wants to keep his application portable cannot make use of all capabilities of the database systems straight away, since these often are not standardized. On the other hand an administrator must not carry out changes in a scheme which provides an application without compromising support and warranty claims. In the end, the access to the database system can simply be refused for him also since it is perhaps a purchased service under constraints and the administrative rights are missing.

DOI: 10.14738/tmlai.25.498
Publication Date: 10th September, 2014
URL: http://dx.doi.org/10.14738/tmlai.25.498

In application and database system environment, this is illustrated in Figure 1. In addition, the registered proxy framework will be presented in this paper. It complements this environment on the network level for new features and it is transparent to the application and the database system.

The following are some works that affect the function of the proxy frameworks, presented (Section 2) and its functions implemented (Section 3.1). An example of index integration (Section 3.2) and a result cache (Section 3.3), illustrates their use and finally some achieved results presented (Section 4).

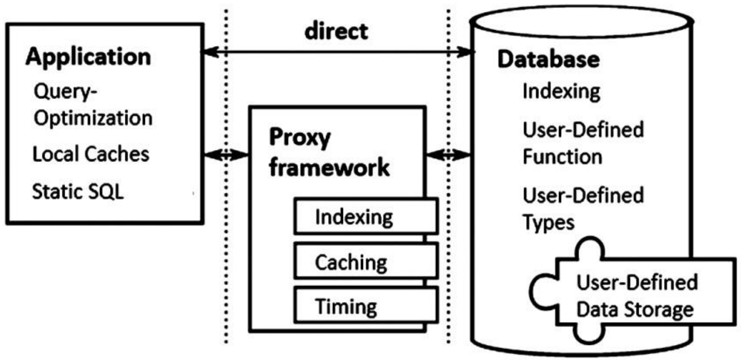

Figure 1: Parts of the existing environment from application and database system that has been extended by our proxy framework. In the proxy framework, some of the potential function modules are registered.

2 Related Work

As mentioned in the introduction, there are primarily two challenges. First, there must be a way to introduce functionalities into multiple database system environments; second, it should be transparently done outside of the database system or application.

Independence of the database system used is offered at the application level through various interfaces, JDBC [7], ODBC [8], OLE DB [9] and ADO (.NET) [10] are highlighted here. Common to all is a way for the application-defined interface. This is, if necessary, shown or passed on directly on a database-specific interface. Such structure (see Figure 2) is always possible if the underlying layer (e.g. Oracle OCI [11]) supports all the features of the overlying layer (ODBC-API).

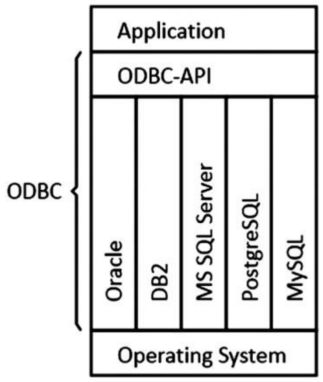

Figure 2: Structure of a system of application, ODBC abstraction layer and database-specific drivers

Furthermore it shall be all about the integration as a proxy in this paper. Also in this area, there are already preliminary works. The concept of a proxy is not new and is designed for database systems, used inter alia by the following developments:

1. *The TDS Protocol Analyzer is a tool that works on the network protocol of tabular data stream (TDS) [12] of Microsoft SQL Server. With this tool, it is possible to intercept the data packets which are sent to record, to analyze, to execute statistical analyses and to monitor vulnerabilities on them. [13]*
2. *GreenSQL is an open source proxy, which is to increase security of MySQL and PostgreSQL, as well as TDS in a commercial version. It scans the communication between the clients and the database system for suspicious requests that might be suitable to exploit security vulnerabilities. Furthermore, it can act as a firewall. [14]*
3. *By means of the Security Testing Framework, invalid packets are generated syntactically for the DRDA protocol used in IBM DB2 (Distributed Relational Database Architecture) [15] to test the robustness of the implementation. [16]*

What all these aforementioned systems have in common is that they work only for selected databases and limited to a few manufacturers. Their area of application is focused on monitoring, testing and access control. Right here, the proxy framework is to apply and offer a possibility, regardless of the database system used, to integrate easily extensible functionality.

A way, how the time performance of the database system can be influenced positively, was presented in [17]. Besides, the index ICIx [18] was requested directly from the application and then the results were used for the expansion of the queries for the database system. This approach is possible only for an application developer.

3 Concept

3.1 Proxy Architecture

Based on the requirements set regardless of the database system used and to be integrated in the network, in the following, the architecture of the proxy framework is presented.

The environment, into which the proxy inserts itself, consists of one or more clients and a database system. The proxy is used in this environment, by all connections which formerly led directly to the database system, are now redirected through the proxy. This then establishes the connection to the database system. Its principle function is to forward data packets received from the client to the database system. Figure 3 illustrates this scenario again.

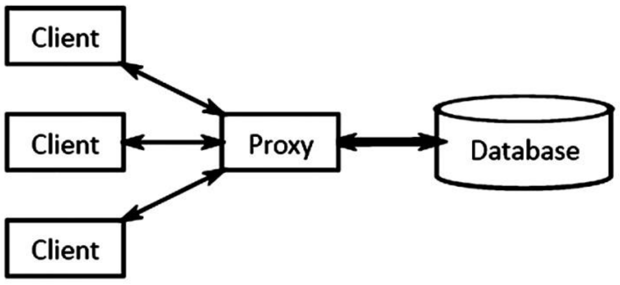

Figure 3: Environment, in which a proxy has been integrated between a database and its clients

Up to this point a pre-control can be already realized by connections (e.g. load balancing, IP access control). Nevertheless, for a further integration of functions, this is not yet enough. A content analysis of the packets to be forwarded is required for this. Client and database system communicate using a protocol, which varies from database system to database system. The proxy framework presented here deals with the database systems IBM DB2, Microsoft SQL Server, and Oracle.

In examining, the database protocols reveal that the sequence of events in the communication between client and the database system consists of the following main steps:

1. *prepare*
2. *bind*
3. *execute*
4. *fetch*

Unlike on the application level, the operations just listed can be combined into composite operations. For multiple bind calls, only a single bind command, which collectedly contains the individual variables, is set off, for example, in Oracle OCI. It is also possible that *prepare*, *bind* and *execute* together are sent in one command. These composite operations can be separated into their individual operations, and then further processed as individual operations.

Based on this observation, the framework provides a set of event-driven callback functions. These exist for each communication direction client → database system (*on_... _request*), as well as for the reverse direction (*on_... _reply*). The following is an overview of functions offered to the user of the proxy framework presented and how they are among other things. All functions have a lot of input parameters that they cannot change and a lot of return parameters, in which any changed values are entered.

- ***on_connect***_...: these functions receive the connection parameters as input. With them it is possible to modify the connection, for example, to stop or to redirect to a different database than the actual target database.
- ***on_prepare***_...: this function prepares requests to the database. The database initially receives the requests and checks the syntactic correctness. At this point, the request can be checked or changed. The change will be important for integrating index in the following. The database responds with a request identifier about the further steps to process.
- ***on_bind***_...: At *prepare* requests with parameters can be provided. These must be bound prior to execution, thus occupied with values. Within this callback function, the parameters can be changed, or, completely held back in the event of a change of the request.
- ***on_exec***_...: Finally it is communicated to the database that everything needed for the execution is present. The call to this callback function is the only time at which the state of the request is defined with its possible binding parameters, because up to this point, parameters can still be bound or all requests will be discarded.
- ***on_fetch*** _...: After the request has been processed in the database, the data can be collected. The client sends a request identifier and then receives the corresponding request and data

response. This callback function makes it possible to create a new result set. This is fixed neither on the size, nor on the type of the original result set.

- ***on_free***_...: Ultimately the data structures that have been created in the database system, as well as in the client, must be cleaned up. All data provided by means of the proxy frameworks for such request can then be released.

These functions are part of an abstract interface. By this if a derivative is provided, then this derivative is called *module*. Modules provide the methods with which the real work is carried out. They are similar to applications that use the ODBC interface, through the callback functions described above, regardless of the database being used. Without loaded module, a standard module is used, which copies the data of the input parameters onto the output parameters. A self-written module inherits all of the methods of these standard modules and needs to overload only the methods that it needed for its function. In the following, two modules are presented, based on this interface, a module for index integration and a result cache module.

3.2 Index Integration

The fundamental integration of an external index in a database was discussed in detail in [19] and [17]. The idea is to provide the database system with the tuple identifiers (TIDs) of the requested records in the query. Database systems can then access the records very quickly and take into account such limitations on the queries in the optimizer by using these TIDs.

Adding the TIDs can be done by using a temporary table that is added to a composite operation for the remaining request, or an IN list which takes over the function of the temporary table, for example. The answer of the index is exactly these TIDs. In an example from [19] as the simple request:

```
SELECT * FROM work WHERE salary > 2000
```

The primary key id (as a TID) extends the request:

```
SELECT * FROM work WHERE salary > 2000
```

AND id IN (4, 18)

Alternatively, a temporary table can be used to circumvent the length restrictions in the database:

```
SELECT * FROM work WHERE salary > 2000
```

*AND id IN (SELECT * FROM tmp_tid_table)*

Methods for *prepare*, *bind* and *execute* must be overloaded. Then the processing for the case of the temporary table is as follows:

1. ***on_prepare_request****: At this point, it can be decided whether the request is to be answered by the index. This query statement can be made solely on the basis of the tables, views, and attributes. If the index is to be used, the query statement is extended by the composite with the temporary table and then passed on. By the analysis of the request, the statically binding parameters are already defined.*
2. ***on_prepare_reply****: The database system returns the request identifier q or an error back as a response. In case of failure, the subsequent method calls are to check this and may make the editing on "pass through".*

3. ***on_bind_request**: If the request q should be processed through the index, calls are set to bind the parameters for dynamically binding parameters. This is important for an index exactly when the binding parameters for the index request are relevant.*
4. ***on_execute_request**: At this point, the request is fully specified. The request for the index is compiled and sent to it after that. The results of the index request are transferred into the temporary table in the database and finally the actual execute of the client is redirected.*

If the use of an IN list is aimed at, some changes arise. For example, the client's original request must be completely suppressed. All binding operations collected must be saved and finally run when *execute* is called. This is insofar difficult, since calling the event-driven methods suppresses *prepare*, also no response from the database releases. There are only two ways to deal with this situation:

1. *Sending a placeholder request whose return is used internally as an identifier for the further operation on the request to be actually indexed.*
2. *An answer from the proxy is sent directly to the client from whom the database has never become aware.*

In both cases, the actual processing is performed in *execute*. The entire operational sequence of communication for the case of the IN list is again presented in Figure 4.

Furthermore, it is possible, depending on the type of the index, to save the complete database query. In this case, the *fetch* method must also be overloaded. It will suppress all messages to the database system and finally the *fetch* generates the results suitably for the client's request from the response of the index.

3.3 Result Cache

Index integration is not the only application of this framework. It is also possible to implement a result cache. Some limitations arise, because integration into the database is not intended. There is no chance to know about data modification if it is done without the proxy. The cache is therefore relevant only for static data. Because queries that ask for the existence of tables, the type of some data records and so on can be systematically viewed as constant, they pose a broad application spectrum.

The cache is configured to specific requests. The result is stored during the first execution and played the same for other identical requests. Even requests with binding parameters can be cached in this way, if for each combination of parameter values, a separate cache entry is created.

For putting into action, the methods for *prepare*, *bind*, *execute* and *fetch* needs:

1. ***on_prepare_requrest**: It is first checked whether it is a request to be cached. If it is the first execution of the request, then a cache entry is created and the request is forwarded to the database. In a repeated execution must be maintained on the possibly binding parameters. The request is therefore retained.*
2. ***on_prepare_reply**: To save the request identifier for later recognition.*
3. ***on_bind_request**: To save binding parameters and hold back.*
4. ***on_bind_reply:** To signal success.*

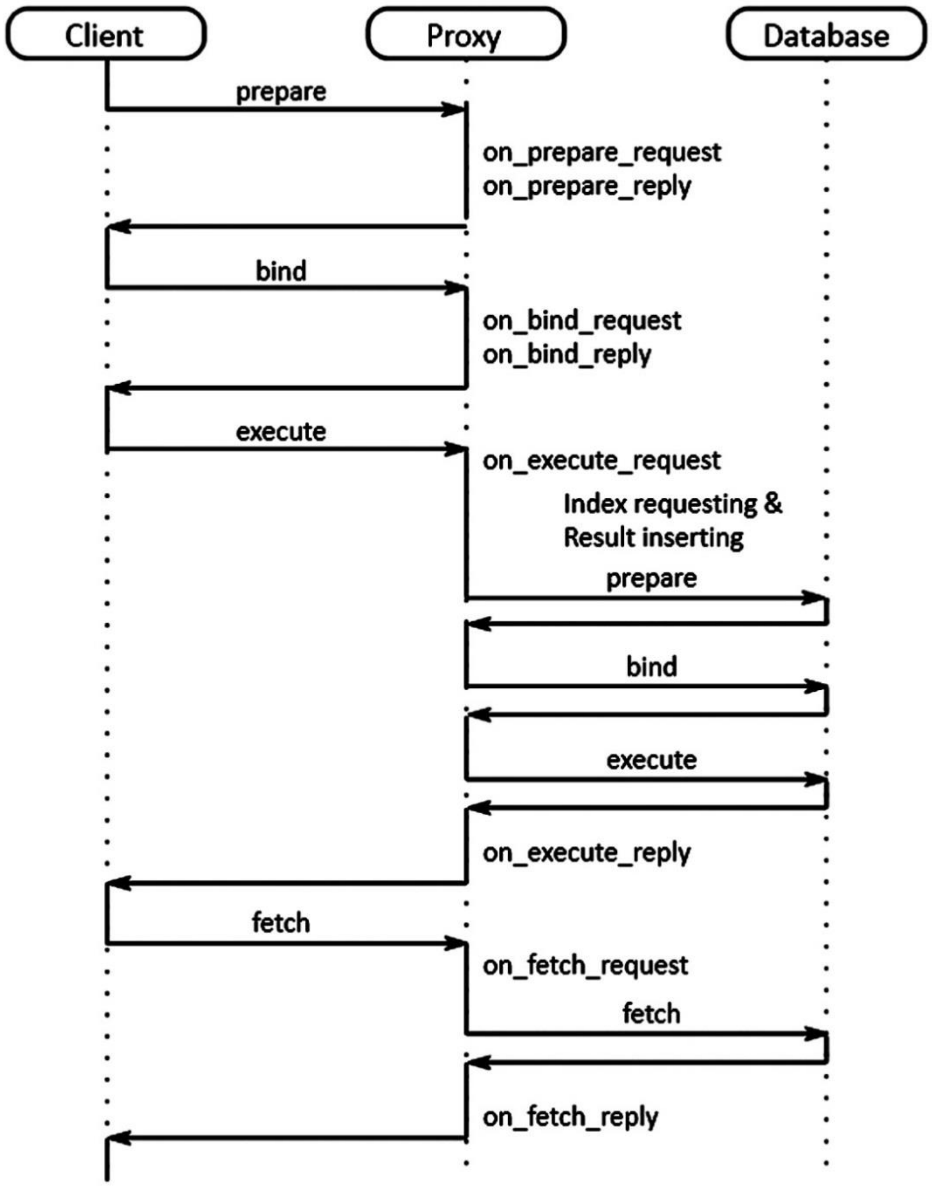

Figure 4: Flow of communications for the index integration using IN-list

1. ***on_execute_request***: First it is checked whether the request including the attached parameters is already in the cache. If this is the case, the request is suppressed to the database. In the case of a new combination of request and parameters, the database is requested.
2. ***on_execute_reply***: The result of the database is returned if requests are not in the cache, otherwise the result from the cache.
3. ***on_fetch_request***: For new requests, the database is queried.
4. ***on_fetch_reply***: For new requests, the results are recorded in the cache. Otherwise it is answered from the cache.

4 Results

The indexing just described has been implemented for the proxy framework. The index usage and the achieved gains are dependent on many factors, such as the database and the query. Thus, in a well-known example [20] an increase in speed by the use of the indexing module for the proxy framework of 32.1 seconds achieved at 0.05 second. The proxy framework required on average 479 microseconds for its part of the processing.

In addition to the time needed by the index itself to answer the requests is already covered in Figure 3, that the times for the network communication come into play twice, because the proxy is interposed. The packages those were previously able to go directly to the database system the way "Client → Database System", must now cover two routes. A scenario with which the proxy framework changed a request statically was tested. The difference in response time at the client with and without proxy was at best 5 milliseconds, the time was still averaged 15 milliseconds.

The processing times within the proxy framework (without modules caused by maturities) are however a few microseconds in the field and play in the face of network latencies (200-300 microseconds measured using **`ping(8)`**) does not matter.

Oracle TNS [21] is an exception. Since no documentation is available for TNS, this protocol had to be previously analyzed. On this occasion, there are several places in the protocol where requests within a TNS packet can be found. This results in an increased workload for this step.

An overview of the times that are needed for certain database protocols within the framework is presented in Figure 5 in the first part of the graph. For this test, requests with predicate part being longer and longer were used. The system used consisted of an Intel Core i5 660 3.33 GHz (two physical CPU cores with hyper threading), 4 GB DDR-1333 RAM under Debian 6.0 (Squeeze). At the beginning, the time required for the parser is not significant and it outweighs the time spent on the protocol.

It is clearly to recognize the increased effort for TNS in the first part of the graph. The relatively strong increase of the graph for DRDA is explained by the fact that the protocol is designed to be very inclusive, which however ensures appropriate overhead time for "normal" scenarios corresponding in demand of computation time.

Furthermore, we have changed result sets as a proof of application possibilities by means of the proxy frameworks. It was found that could not only insert the same data types, but also, depending on the client used, could create completely new result sets and types. At Clients, which evaluate the result at run time, completely different data could be transmitted, than were requested. However, if the expected data type is not checked, this is not possible. In this way, the simple request "**`SELECT 2 FROM dual`**", the result is "2", we were able to return a completely new table in sqlplus [22], which consisted of several attributes per tuple.

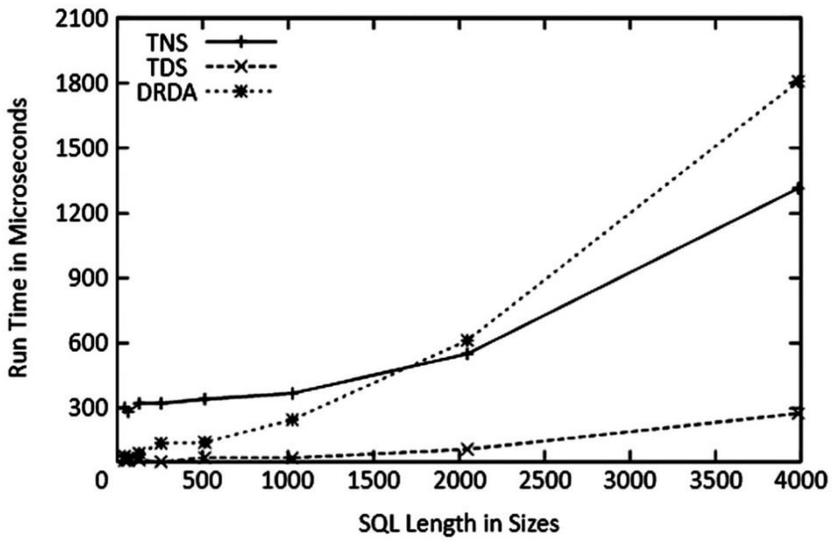

Figure 5: Processing times within the proxy framework for various database protocols and various sizes of the contained SQL queries

In addition, a function was implemented for the time analysis which measures the time of the first *prepare* up to the last response on *fetch*. If there is no access to an evaluation functionality of the database system, this is very useful. In our case, it is used to determine index candidates.

Already in section 3.1, the restriction is introduced by encryption, to which the proxy framework is subject. The encryption is end-to-end and cannot be simply bypassed. If this is desired, it must each implement an end and starting point for the encryption and they will therefore be repealed for the modules within the proxies in the proxy framework.

5 Outlook

Beside the presented proxy framework, we deal with other ways to integrate functionalities into different database systems. This also applies that a change of the database system or application is to be avoided. For integration on library level on client side, the event-driven approach is not good as there exists a way more elegant one. So no placeholder query is needed for the IN-list variant of the index integration e.g., as the return code from *prepare* can signal the successful processing. A standardization of these two so-called integration points is the goal of our further work.

As already described in [19], the challenge also positions itself with this integration variant to recognize data-changing operations, if not the whole data traffic is handled via the proxy framework described here. On this issue, which is not only specific to the proxy framework, we will discuss in the future.

REFERENCES

[1]. IBM Deutschland GmbH. DB2 Spatial Extender und Geodetic Data Management Feature – Benutzer- und Referenzhandbuch, July 2006.

[2]. C. Murray. Oracle Spatial Developers Guide, 11g Release 2 (11.2). Oracle, Dec. 2009.

[3]. IBM. SQL Reference, Volume 1. IBM Corporation, Nov. 2009.

[4]. B. Rich. Oracle Database Reference, 11g Release 2 (11.2), Sept. 2011.

[5]. K. Stolze and T. Steinbach. DB2 Index Extensions by example and in detail, IBM Developer works DB2 library. Dec. 2003.

[6]. E. Belden, T. Chorma, D. Das, Y. Hu, S. Kotsovolos, G. Lee, R. Leyderman, S. Mavris, V. Moore, M. Morsi, C. Murray, D. Raphaely, H. Slattery, S. Sundara, and A. Yoaz. Oracle Database Data Cartridge Developers Guide, 11g Release 2 (11.2). Oracle, July 2009.

[7]. R. Menon. Expert Oracle JDBC Programming. Apress, 2005.

[8]. International Standard for Database Language SQL -Part 3: Call Level Interface, 1995.

[9]. J. Blakeley. In Compcon '97. Proceedings, IEEE, title=Universal data access with OLE DB, pages 2–7, 1997.

[10]. D. Sceppa. Microsoft Ado. Net (Core Reference). Microsoft Press, 2002.

[11]. J. Melnick. Oracle Call Interface Programmer's Guide, 11g Release 2 (11.2). Oracle, Oct. 2009.

[12]. Microsoft Corporation. Tabular Data Stream Protocol, Jan. 2013.

[13]. L. Guo and H. Wu. Design and implementation of TDS protocol analyzer. In Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on, pages 633–636, 2009.

[14]. http://www.greensql.com, 2013.

[15]. The Open Group. DRDA V5 Vol. 1: Distributed Relational Database Architecture, Aug. 2011.

[16]. M. Aboelfotoh, T. Dean, and R. Mayor. An empirical evaluation of a language-based security testing technique. In Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research, pages 112–121. ACM, 2009.

[17]. S. Leuoth, A. Adam, and W. Benn. Profit of extending standard relational databases with the Intelligent Cluster Index (ICIx). In ICARCV, pages 1198–1205. IEEE, 2010.

[18]. O. Görlitz. Inhaltsorientierte Indexierung auf Basis künstlicher neuronaler Netze. PhD thesis, 2005.

[19]. Adam, S. Leuoth, and W. Benn. Minimal-Invasive Indexintegration – Transparente Datenbankbeschleunigung. In I. Schmitt, S. Saretz, and M. Zierenberg, editors, Grundlagen von Datenbanken, volume 850 of CEUR Workshop Proceedings, pages 83–87. CEUR-WS.org, 2012.

[20]. Council, Transaction Processing Performance. TPC-H benchmark specification. Published at http://www.tcp.org/hspec.html, 2008.

[21]. D. Litchfield. The Oracle Hacker's Handbook: Hacking and Defending Oracle. John Wiley & Sons, Inc., New York, NY, USA, 2007.

[22]. S. Watt. SQL*Plus – Users Guilde and Reference, Release 10.2. Oracle, June 2005.