RESEARCH REVIEW JOURNALS
Volume-04
Issue-06
June-2019

ISSN: 2455-3085 (Online)
RESEARCH REVIEW International Journal of Multidisciplinary
www.rrjournals.com[UGC Listed Journal]

Implementation and Testing of SBQL Object-Relational Wrapper Supporting Query Optimisation

[1]Enam Ahmad & [2]Dr. Manish Kumar

[1]Research Scholar, Department of Computer Science, (Under P.G. Dept. of Mathematics), Magadh University, Bodh-Gaya (India)
[2]Associate Professor, P.G.Dept. of Mathematics, A.N. College, Patna (India)

ARTICLE DETAILS

Article History
Published Online: 12 June 2019

Keywords
prototype test, relational database ,object oriented updateable view, virtual object , virtual repository schema, effective query optimisation, Sample optimisation results.

ABSTRACT

This paper presents design, implementation and prototype tests of an object-oriented wrapper enabling virtual integration of relational databases to an object- oriented database. The integration process is transparent - the users need not be aware that data are delivered by a relational database. The object-oriented model consists of virtual objects that can be in turn arbitrarily transformed by object-oriented updateable views, according to the virtual repository schema. Virtually transformed relational data can be combined in queries with purely object-oriented data and with data from other wrapped resources. Due to this implementation need more attention for effective query optimisation. The described prototype employs dedicated query rewriting methods so both object-oriented and relational query optimisers can work together. Sample optimisation results and comparisons are presented.

1. Introduction

Object relational wrappers or mappers,ORM for integrating relational databases within object-oriented environments are developed for about fifteen years. A commonly recognized reason for this effort is impedance mismatch between relational databases (SQL) and object-oriented models having specific programming constructs. The most recognized products of this development are JDO, EJB, Top link, Hibernate and other solutions for Java and .Net, including native queries [1].

So many uncertainty related usefulness of the goals that are pursued by ORM-s. The doubts can be summarised into two points: (1) limitations of an object-oriented model that are forced by the fact that the target data is relational; (2) performance penalty implied through ORM in case of huge relational databases. Both these doubts are closely related to each other. An attempt to restore an object-oriented model (e.g. originally developed in UML) from a target relational database requires non-trivial mappings which correspond to sophisticated views in databases and to the well-recognized view updating problem. This challenging conceptual problem would be solved for a less sophisticated case, but the performance problem still exists. It could be impossible or, atleast, very challenging to translate all the requests (including updates) to an object-oriented schema into SQL in such a way that SQL internal optimizers can work efficiently. The mapping that participates in this translation causes many limitations of the final SQL statements, up to the case when these statements have the trivial form *select * from Relation* and the results of the statements are processed sequentially by cursors or iterators. This type of SQL statements gives no chance to SQL optimizers. Processing big relational databases without query optimization is unacceptable for majority of business applications.

In effect, mapping between a relational schema and an object-oriented schema, as well as mapping between object-oriented queries and SQL, must be straight forward or trivial for huge relational databases. Actually, the mapping is usually isomorphic, upto secondary Syntactic differences. In this way an object-oriented model is still a “slave” [2] of the relational model. Lots of reasons, including seamless transition from an object-oriented analysis and design model (e. g. in UML) into object-oriented implementation model (e.g. in Java) such a “slavery” is contradictory to fundamental tenets and promises of the object-orientedness as a software engineering paradigm.

Thus the fundamental question how to get full freedom concerning the mapping between relational and object-oriented oriented schemas without compromising performance?

In our investigations within the European project eGov Bus [3] we have struggled with the problem. Our task in this project concerned creating generic software for making virtual repositories (VR-s). A VR has to integrate virtually existing (legacy) data and service resources, including relational databases, XML and RDF files, data available via Web Services, etc. An essential assumption of a VR is reducing the global conceptual data and services schema to the conceptual (perhaps minimal) form that is required by the given global applications, Hence, a VR has to achieve many forms of data organization and access transparency, including distribution, heterogeneity, data representation, redundancy, replication and fragmentation transparencies. Ideologically, a VR reminds a CORBA bus, but with several conceptual changes that include richer (UML-like) data model with explicit nested collections, access to virtual objects through a query language rather than via low-level programming, the possibility to build programming and database abstractions (procedures, functions, classes, methods view, etc.), flexibility concerning external schemata for client applications, and others. The detailed description of the software for making virtual repositories (VR-s) is the matter of other documents (see [4]). In this paper we focus only on the part that is devoted to mapping relational databases to an object-oriented schema. All the research and development is based on the object database for Rapid Application development (ODRA)

system[4], the object-oriented query and programming language Stack-Based Query Language (SBQL) [5] and the other new concept of virtual updatable views [4,6,7] that extends the capabilities of SBQL. Just due to the power of our views we have started to describe the ODRA system as Virtual Repository Management System (VRMS).

In contrast to all past proposals concerning views (including SQL), our views are defined by programming functions having the full algorithmic power. The power concerns the mapping from a stored data into virtual objects, as well as the reverse mapping of updating operations addressing virtual objects into updating operations addressing the stored data (thus our idea reminds in place of trigger views of SQL, but it's based on different assumptions). This power causes, SBQL view make it possible to get effects that so far have not even been considered in the database domain.

Besides SBQL and virtual object-oriented views our idea is based on the query modification technique and an architecture that will be able to detect in a query syntactic tree some patterns that can be directly mapped as optimizable SQL queries. The patterns match typical optimization methods htat are used by the SQL query optimizer, in particular, rewriting, indices and fast joins. Due to JDBC currently supported RDBMS include Axion, Cloudscape, DB2, DB2/AS400, Derby, Firebird, Hypersonic, Informix, InstantDB, Interbase, MS Access, MS SQL, MySQL, Oracle, Postgres SapDB, Sybase and Weblogic.

The idea behind wrapper is that a relational database is treated as a primitive object-oriented database, where each tuple is considered an object. Then, on such a primitive object-oriented database the developer of a VR can define virtual views that map it to the given object-oriented model assumed by the global object- oriented schema. SBQL queries invoke the object-oriented model, thus the relational database structure is fully transparent for its users. Due to the power of SBQL views, any complex mapping between a relational schema and an object-oriented canonical schema is feasible. The processing of SBQL queries addressing a virtual object base is done by the following steps:

- SBQL query is compiled and then its abstract syntax tree (AST) is produced;
- Each node of AST that contains a view invocation is substituted by the view AST; which is known as query modification method;
- In the result we obtain a large AST representing an SBQL query with no view invocations and addressing the relational database. This query address is just the primitive object database i.e. 1:1 compatible with the relational database. It is first optimized by the SBQL engine by removing dead subqueries, factoring out independent subqueries from loops, etc,;
- The resulting syntactic tree can't be fully mapped to the SQL, because SBQL is more powerful than SQL and SBQL queries can refer to a local environment which is unavailable for SQL. Hence, the tree is traversed in-order to discover largest subtrees that are 1:1 compatible with SQL queries;
- Such subtrees are then mapped into SQL code using the JDBC interface;
- After that the tree is compiled to the SBQL machine code (byte code) and then executed. The results from JDBC invocations are converted to the SBQL format and stored at SBQL stacks.

Benchmarks have shown that this algorithm behaves quite well and is able to utilize almost all native SQL optimization methods.

In this paper we also omit the brief details of SBA, SBQL, ODRA and the VR software due to dozens of other sources; see [4,5]

The concept of wrappers and mediators [8] to heterogeneous distributed data sources was introduced as a vision and an indication for developing future information systems for next years. The perspective and the technique presented in [8] had several implementations, e.g. Pegasus [9], Amos [10], Amos II [11] and DISCO [12]. The mediation approach is crucial for the described wrapper. Even so, there exist many other approaches aiming to build object relational data (e.g. XPERANTO [13], XTABLES [14]), also with data updating capabilities some experimental applications of RDF and SPARQL (e.g. [15]).

The rest of the paper is arranged as follows, section 2 deals with the implemented wrapper architecture and query processing. Section 3 deals with the query optimization, which is explorer with examples. It also presents optimization results. Section 4 represents our conclusion and Section 5 References.

2. Wrapper Architecture and Query Processing

Figure 1 (Fig. 1) Explain the wrapper Section within a virtual repository. An ODRA resource denotes any data resource providing an interface capable of executing SBQL queries and returning SBQL objects an their results (a local query optimization should be performed also, if possible). The nature of such a resource is in torrent, as only the mentioned capability is important. In the simplest case, where a resource is an ODRA database, its interface has direct access to a data store and its identical with an ODRA database engine. However, as the virtual repository aims to integrate existing business resources, whose models are mainly relational ones, an interface becomes much more complicated, as there is no directly available data store – SBQL result objects must be created dynamically basing on results returned from SQL relational queries evaluated directly in a local RDBMS.

Such a case (most common in real-life applications) forces introducing an additional middleware, i.e. a wrapper realized in the client-server architecture (Fig.1), which assures simplicity of implementation, maintenance, protability and distribution. A regular ODRA database can be extended with as many wrappers as needed (e.g., for relational or somistructured data stores, RDF resources, etc.) and plugged into any resource model without any lost of its primary performance.

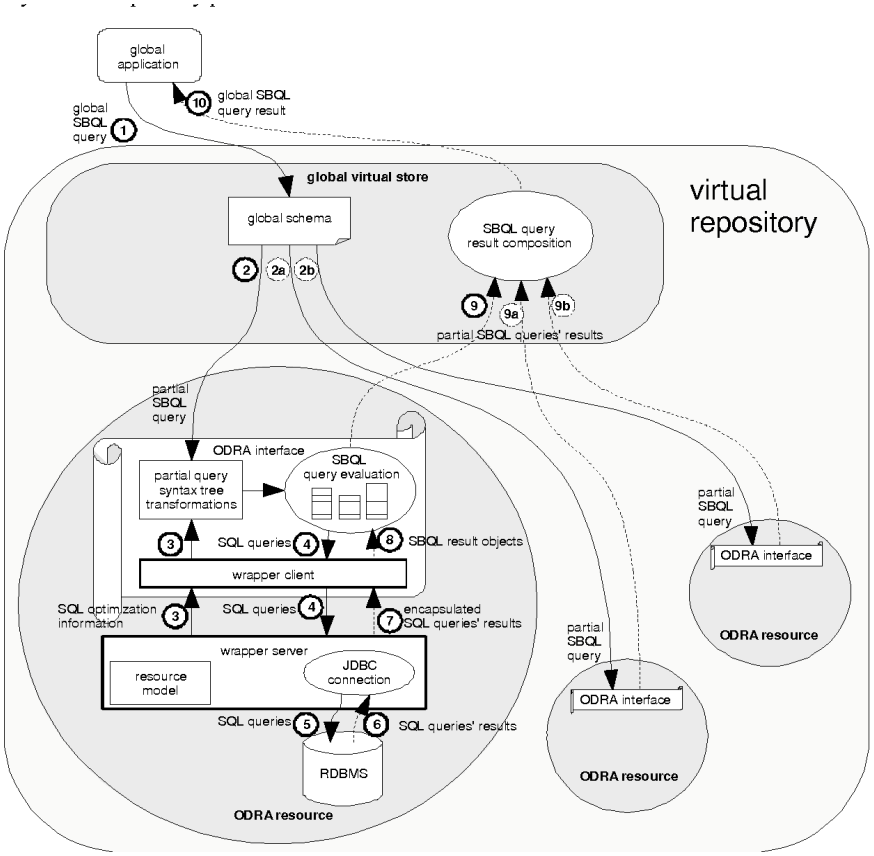

Fig. 1. Virtual repository and wrapper query processing System.

A query assessment process with application of the wrapper is described in details in Fig.1. One of the main role of the applications to sends a query (arrow 1). This query is expressed with SBQL, as it refers to the business object oriented model available to Top-level users. According to the global schema and its information on data fragmentation, replication and physical location (obtained from integration schema and from the global index), the query is decomposed into subqueries (partial queries) and sent to appropriate resources. This stage is realized with arrows 2, 2a and 2b. A notion “partial query” is general, as in some cases each resource-oriented query can be the same, however in most situations partial queries are different (issues of data fragmentation and integration are discussed and the possible solutions presented in [16, 17]). Query processing corresponding to arrows 2a and 2b is out of the scope of the paper as those resources are represented here as black boxes.

The imperfect query aiming at our relational resource is further processed with a resource's ODRA interface. As above mentioned, a local interface does not have its physical data store; it can only retrieve required data from its RDBMS on the fly. First, the interface performs query optimisation. Besides efficient SBQL optimisation rules applied at any resource's interface, here one can also transform queries so that powerful native SQL optimisers can work and amounts of data retrieved from the RDBMS are acceptably small (just matching selection condition). Relational maximum efficiency information (indices, cardinalities, primary-foreign key relationships, etc.) is provided by the wrapper server's resource model (arrow 3) and appropriate SBQL query syntax tree transformations are performed. These transformations are based on finding in the SBQL syntax tree patterns corresponding to SQL-optimisable queries. A proper tree branches are substituted with calls to *run SQL* expression with optimisable SQL queries. Their evaluation is fast and efficient at the resource and returned results are acceptably small. Even though,some queries may not be much optimised due to their nature excluding efficient transformation to optimal forms.

Once syntax tree transformations are completed, the interface starts a regular SBQL query evaluation. Whenever it finds an *execute SQL* procedure, its SQL query is sent to the wrapper server via the client (arrow 4, the client passed SQL queries without any changes). The server executes SQL queries as a resource client (JDBC connection), arrow 5, and their results, arrow 6, are encapsulated and sent to the client (arrow 7). Subsequently, the wrapper client creates SBQL result objects from results returned from the server and puts them on regular SBQL stacks for further evaluation (arrow 8).

Having finished local evaluation, the interface sends its *partial result* upwards (arrow 9), where it is combined with results returned from other resources (arrows 9a and 9b) and the global query result is composed (depending on fragmentation types, redundancies and replication). This result is returned to the global application (arrow 10).

The procedure of wrapping relational schemata is not crucial for understanding the query rewriting examples provided below, still it explains some naming conventions.

The first step consists of creating an ODRA metabase mapping one to one the relational schema. Over this metabase automatically generated basic wrapper views and virtual pointer objects created for primary-foreign key pairs. During this automatic view generation stage, every “relational” name is given a corresponding object-oriented view. The resulting view-based schema (realising the SBA AS0 model) is ready for direct querying or integration to the virtual

repository via arbitrarily complex integration and contributory views.

The example of the mapping procedure is illustrated on the test relational schema used for the following optimisation examples. This schema consists of three tables (Employees, Departments and Locations) related by primary-foreign key constraints populated with random data. Each table has its primary key column (**id**); there are also non-unique indices on Employees' surnames and salaries, Departments' names and Locations' names.

The wrapping procedure is shown in the following below figures. Fig.2 explains the plain RDBMs schema with tables and their relations shown.

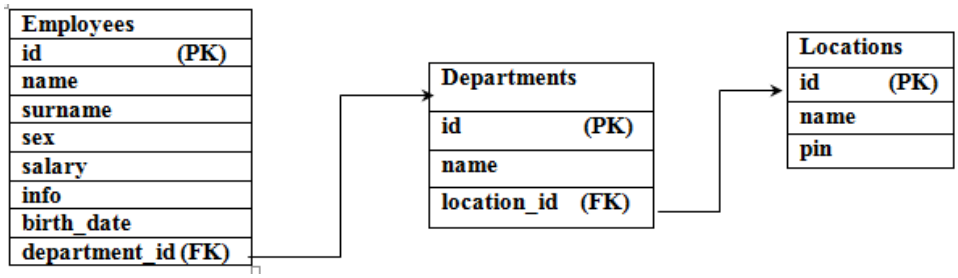

Fig. 2. Test relational schema

The result of the primitive wrapper mapping is depicted in Fig.3 – three complex metaobjects are created in ODRA metabase.

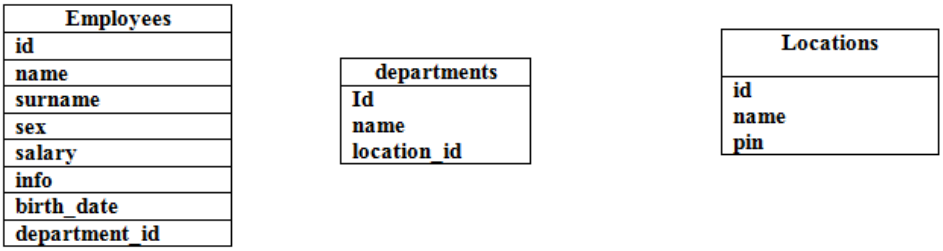

Fig.3. Object-oriented schema mapping one-to-one relational tables and columns.

These metaobjects are then covered with views realising the M0 SBA model (Fig.4) virtual pointers are created for foreign key columns to realize the wrapped schema integrity.

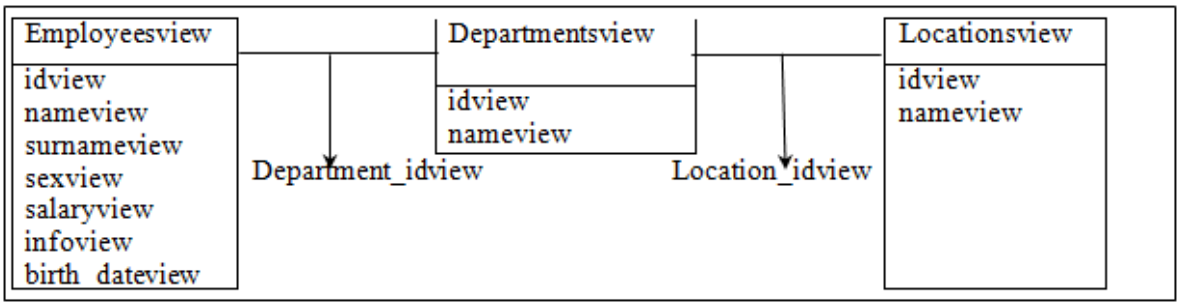

Fig. 4. Object-oriented schema covered by automatically generated wrapper views

3. Query Optimisation Methods, Examples and Results.

Besides reliable and transparent integration procedures, the main goal of the described solution is how to utilize a native powerful SQL optimizer in order to push query evaluation down to the wrapped resource, to minimize data throughput over network and to prevent un useful ODRA store space consumption in all known RDBMSs the optimizer and its particular structures (e.g., indices) are transparent to the SQL users. A naïve implementation of a wrapper causes that it generates primitive SQL queries such as select * form tablename, and then, processes the results of such queries by SQL cursors. Hence the SQL optimizer has no chance to work. This problem has been solved successfully and most of query result processing is performed by a RDBMS under a wrapper. The solution is based on SBQL, virtual object-oriented views defined in SBQL, query modification methods, and dedicated optimization methods able to detect in a SBQL query syntactic three some patterns that can be directly mapped as optimiseable SQL queries. The patterns match typical optimization methods that are used by the SQL query optimizer, in particular, indices and fast joins.

The substantial step allowing the wrapper to access any SBQL semantics is macro-substituting view definitions for their names so that only "relational" names remain (the view rewriting and query modification procedures); the process is executed automatically as only such a form of a query can be correctly recognized and transformed by either wrapper rewriter or optimizer. Currently the wrapper is capable of performing its effective optimization on most queries however some SBQL expressions (e.g. procedure calls) cannot be transformed to semantically equivalent SQL structures. In

such cases, in order to make query evaluation possible, relational data is retrieved to ODRA with no SQL optimization and further processed here.

The assumed optimization procedure relies on common rules valid for both relational and object-oriented database systems i.e. perform selections and projections as early as possible in order to minimize necessary processing. IO operations (here they are substantial in the distributed system architecture) and storage. The wrapper optimizer examines the query syntax tree for statements that can be transformed into SQL- optimisable queries (being subqueries of the SBQL query). The rewriteable expression search order is: aggregate functions, joins, where conditions, finally single-table expressions (just to determine required projected columns). This order is introduces so that the possibly largest part of the SBQL syntax tree can be replaced with the corresponding SQL query. For example, some join expression can contain conditions expressed with where clause. If where clauses were transformed before joins, the query tree would become "corrupted" and no rewritable join patter would be found afterwords. This happens because the execute SQL expression encapsulates the semantics of the substituted SBQL subquery/tree branch, which is enough for its correct evaluation but the reverse process would require semantic analysis of the SQL query. This is not implemented as it is currently unnecessary.

The examples presented below are summit on the test schema introduced earlier with the wrapping procedure description. A data used for tests are randomly generated according to some predefined distribution; therefore the optimization results may very for different test databases. Tests' results become more repeatable if the population of the Employees table grows up and its data distribution is closer to the assumed pattern. Here presented optimization results are average values for 10 subsequent measurements for populations of 10, 100 and 1000 employees. The polts represented in the log liner scale; both the evaluation time ratio and ODRA store occupation are given.

As declared above, the wrapper optimizer searcher for the largest syntax tree branch possible for the optimization, the result returned from the resource is indistinguishable from other objects so the ODRA query assessment engine processes it, in its regular stack based manner. This feature enables embedding execute SQL expressions in larger SBQL queries with preservation of all original typing information. These cases are not considered in the examples due to their complexity and little impact on optimization results.

Example 1

Let's consider a query target to retrieve surnames and names of employees earning more than 1100. In SBQL, correspondingly to the test schema, it can be expressed as:

(Employeesview where salaryview>1100). (surnameview, nameView);

The query modification step converts it into the following form, where only "relational" names exist (there can also appear some auxiliary SBQL names):

((employees where ((salary.deref(_VALUE))>(real)(1100))). (surname,name))

The query in such form is prepared for wrapper optimization (or only simple rewriting in the worst case). The optimization procedure results in:

execsql("select employees.surname,employees.name from employees where

(employees.salary>1100)" , "<resultpattern>" ,"wrapper")

Here, in a simple query, only a single fired SQL expression appears. It prevents the SQL query string, some result pattern. The SQL query string is forwarded to a wrapper named wrapper (it is specified as the most recent (last) execute SQL parameter as they can be various wrappers mixed in a single query), carry out optimally in the wrapped resource, the result is returned to the wrapper client and corresponding ODRA objects are created. The SQL query should be optimally evaluated by the wrapped resource engine as direct selection criteria's and projected columns are given.

Example 2 Below we present a query with no optimization but simple rewriting only. Such a case holds if a SBQL query invokes some constructs having no counterparts in SQL.

((**execsql**("selectEmployees.info,Employees.Depatment_id,Employees.surname,

Employees.salary,Employees.id,employees.sex,employees.name,employees.birth_date from employees","<resultpattern>","wrapper")where((salary.deref(_VALUE))>(real)(1100))).(surname,name))

The generated string of an SQL query retrieves all the record form the employees table and actual selection and projection are performed by ODRA.

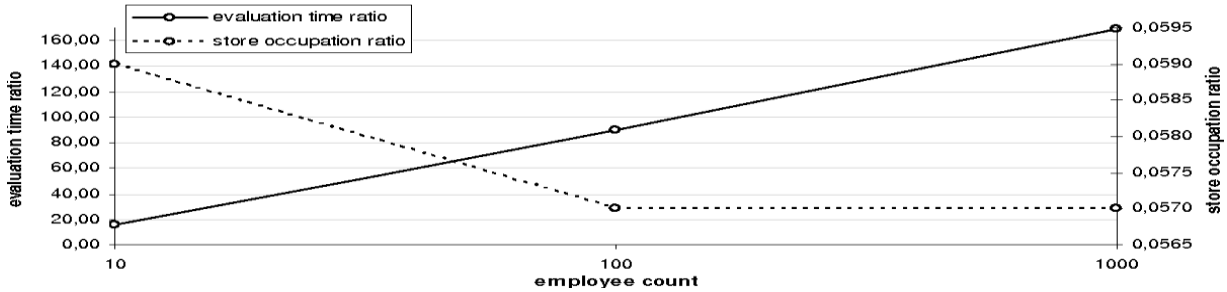

Fig. 5. Optimisation results for **Example 2** examples.

Now let's introduce a more complex query combining more relational tables. It aims to *retrieve surnames of employees and cities their departments are located in* (join performed by *means of a virtual pointer),*

(employeesView as e join.department_idView.departmentsView as d join d.location_idView.locationsView as). (e.surnameView, l.nameView);

After views' optimisation the query is:

((((employees) as e join ((((e . (department_id) as _employees_department_id) . ((departments where ((id .deref(_VALUE))=deref((_employees_department_id. _VALUE)))))as departmentsView).departmentsView)) as d) join ((((d .(location_id) as _departments_location_id). ((locations where ((id . deref(_VALUE)) = Deref ((_departments_location_id . _VALUE))))) as locationsView).locationsView)) as l).((e . surname),(l.name)))

And after the wrapper optimisation:

execsql ("select employees.surname,locations.name from employees, locations,departments where ((departments.id = employees. department_id) AND(locations.id= departments.location_id))","<result pattern>","wrapper")

The created SQL query can be optimised and efficiently executed in the wrapped relational database since it contains join conditions expressed with primary-foreign key constraints.

Here is a non-optimised form of the query:

((((**execsql**("select employees.info, employees.department_id, employees.surname, employees.salary, employees.id, employees.sex, employees.name, employees.birth_date from employees", "<result pattern>", "wrapper")) as e join (((e.(department_id) as _employees_department_id) .(**execsql**("select departments.name, departments.location_id, departments.id from departments", "<result pattern>","wrapper")where((id .deref(_VALUE)) = deref ((_employees_department_id . _VALUE)))))) as d) join (((d . (location_id) as _departments_location_id) . (**execsql**("select locations.id, locations.name from locations", "<result pattern>", "wrapper") where ((id.deref(_VALUE)) = deref ((_departments_location_id . _VALUE)))))) as l) . ((e.surname) , (l.name)))

It retrieves all data from the joined tables (although no projection is required for the *departments* table as it is used only for the join operation). Join calculation and the final result projection are performed by ODRA, again.

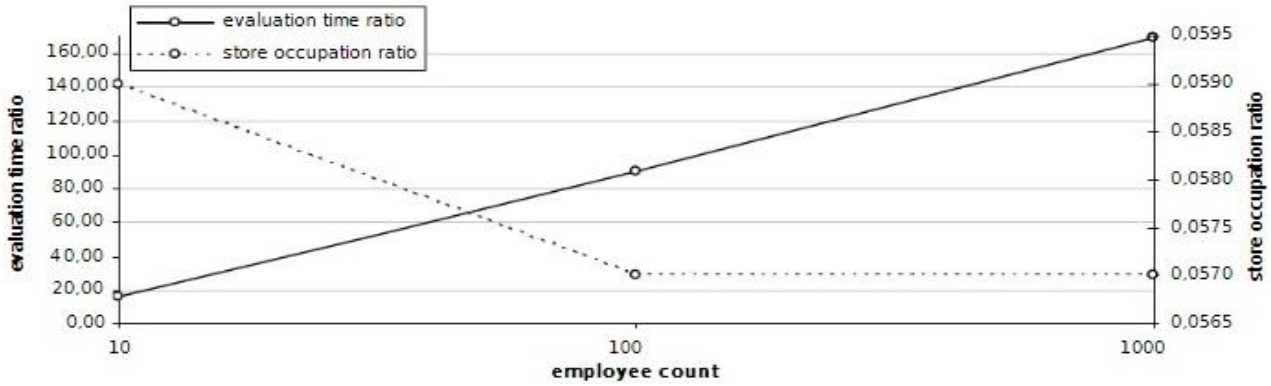

Fig. 6. Optimisation results for Example 2.

Example 3 The last presented query example aims to *retrieve the sum of salaries of employees named Raj working in Lord city* (direct navigation through virtual pointers):

sum((employeesView where surnameView = "Raj" and department_idView. departmentsView.location_idView.locationsVi ew.nameView = " *Lord* ").salaryView);

After the view rewriting:

sum((((employees where(((surname.deref(_VALUE)) = "Raj") and (((((((((department_id) as _employees_department_id.((departments where((id.deref(_VALUE)) =deref ((_employees_department_id . _VALUE))))) as departmentsView).departmentsView). (location_id) as _departments_location_id).((locations where ((id . deref(_VALUE)) = deref((_departments_location_id . _VALUE))))) as locationsView).locationsView).name) . deref(_VALUE)) = " *Lord* "))) .salary).deref(_VALUE)))

And after the wrapper optimisation:

execsql("select sum(employees.salary) from employees, locations, departments where ((employees.surname = 'Raj') AND ((locations.name = ' *Lord*') AND ((departments.id = employees. department_id) AND (locations.id=departments.location_id))))", "<result pattern>","wrapper")

Only a single SQL query is executed and selection, projection and sum calculation are pushed to the wrapped database (ODRA receives only a number being the final result).

The worst case is when the query is non-optimised:

sum((((**execsql**("select employees.info,employees.department_id,

employees.surname, employees.salary, employees.id, employees.sex,employees.name,employees.birth_date from employees", "<result pattern>","wrapper") where(((surname .deref(_VALUE))="Raj") and(((((((department_id)as _employees_department_id. (**execsql**("select departments.name, departments.location id,departments.id from departments", "<result pattern>","wrapper")where((id.deref(_value))= deref((employees_department_id.value))))).(location_id)as_departments_location_id).(**execsql**("selec t locations.id, locations.name from locations", "<result pattern>","wrapper") where ((id . deref(_VALUE)) = deref ((_departments_location_id . _VALUE))))).name).deref(_VALUE)) = "*Lord*"))).salary). deref(_VALUE)))

Similarly as in the previous case, data from all the tables to be joined are retrieved, joins are evaluated by ODRA according to the given conditions, finally selection and projection are applied and the sum of salaries is calculated.

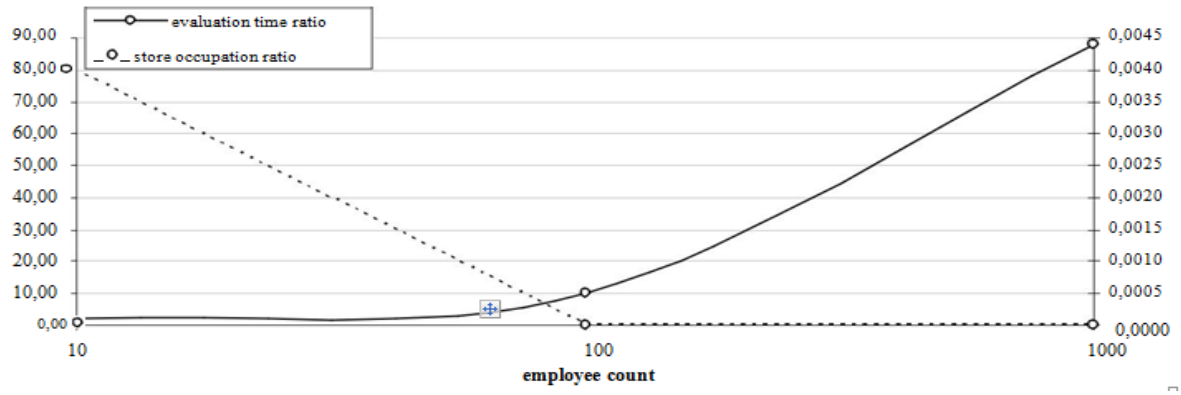

Fig. 7. Optimisation results for example 3

4. Conclusions

The implemented prototype has shown that the presented Method is feasible, functional and efficient. The main assumptions concerning transparency –no data materialization and efficient query optimization are successfully achieved. Sometimes full data retrieval is required if some query cannot be optimized. Even in such case it is evaluated correctly. However even such conditions are not hopeless, as before generating SQL Queries very efficient SBQL optimization can be applied, e.g., factoring out independent subqueries [18,19] or additional indices. The presented diagram and methods are very flexible for wrapping not only relational databases. Recently we also work on virtual integrating other, non-relational resources, in particular, oriented towards RDF.

In this topic we do not dicuss query rewriting procedures for multi-wrapper(transferring to more than one wrapper) and mixed queries(i.e. combining wrapped objects and pure ODRA objects).

References

1. Adamus R., Kowalski T.M., Wislicki J. (2011) Implementation of Updateable Object Views in the ODRA OODBMS. In: Meersman R. et al. (eds) On the Move to Meaningful Internet systems: OTM 2011. OTM 2011. Lecture notes in Computer Science, vol 7045. Springer, Berlin, Heidelberg, http://doi.org/10.1007/978-3-642-25106-1_23.
2. Cybula P., Subieta, K.: Query Optimization through cached queries for object-oriented query Language SBQL. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorny, J., Rumpe, B. (eds.) SOFSEM 2010, LNCS, vol. 5901, pp. 308-320, Springer, Heidelberg (2010)
3. eGov-Bus project, http://www.egov.-bus.org/web/guest/home,2008.
4. virtual Repository Management System ODRA, http://www.sbql.pl/virious/, 2008.
5. Subieta K.: Stack-Based Architecture and Stack-Based query language, http/:www.sbql.pl/, 2008
6. Kozankiewicz H., Leszczylowski J., subieta K.: implementing mediators through virtual updateable views, engineering federated information systems, proceedings of the 5th workshop EFIS 2003, july 17-18-2003, coventr, UK, pp. 52-2.
7. barbosa, D.M.J., Cretin. J., foster. N., Greenberg. M., Pierce, B.C.: Matching lenses: alignment and view update. In: ACM SIGPLAN international conference on functional programming (ICFP). Pp. 193-204. ACM, USA (2010)
8. Wiederhold G.: Mediators in the architecture of future information systems. IEEE Computer, 25(3), 1992, pp. 38-49.
9. Ahmed R., Albert J., Du W., Kent W., Litwin W., Shan M-C.: an overview of Pegasus. In: proceeding of the workshop on interoperability in multi database systems, RIDE-IMS' 93, Vienna, Austria, 1993.
10. K.Ramachandra, R. Guravannavar, and S. Sudarshan. Program analysis and transformation for holistic optimization of data base applications. In Proc. SOAP workshop, pages 39-44, 2012.
11. Risch T., Josifovski, V., Katchaounov, T.: functional data integration in a distributed mediator system. In functional approach to computing with data, P. Gray,

L. Kerschberg, P. King, and A. Poulovassilis, Eds. S[romger, 2003.

12. Tomasic A., Raschid L., Valduriez P.: scaling access to heterogeneous data sources with DISCO. IEEE Transactions on Knowledge and Data Engineering, Volume 10, pp 808-823, 1998.
13. Carey M., Kiernan J., shanmugasundaram J., Shekita E., Subramanian S.: XPERANTO: A middleware for publishing object-relational data as XML documents.
14. Funderburk J.E., Kiernan J., G., Shanmugasundaram J., Shekita E., Wei C. : XTABLES: bridging relational technology and XML, IBM systems journal, 41, No. 4, 2002.
15. Perez de Laborda C., Conrad S.: Bringing relational data into the semantic Web using SPARQL and relational. OWL, data Engineering workshops, 2006. Proceedings. 2006, pp. 55-55.
16. Kuliberda K, Adamus R., Wislicki J., Kaczmarski K., Kowalski T., Subieta K.; Autonomous layer for data integration in a virtual repository, on the move to meaningful internet systems 2006, 3rd international conference on Grid computing, high-performance and distributed applications (GADA' 06), Springer 2006 LNCS 4276, Montpellier, France, 2006, pp. 1290-1304.
17. Kuliberda K., Adamus R., wislicki J., Kaczmarski K., Kowalski T., Subieta K.: A Generic proposal for a Transparent integration of distributed data by an autonomous layer in a virtual repository, mulitiagent and grid systems journal (MAGS), no- 4, vol 3, 2007 (to appear)
18. Bleja, M., Kowalski, T.M. Adamus, R., Subieta, K.: Optimization of object oriented queries involving weakly dependent Subqueries. In: Norrie, M. C., Grossniklaus, M. (eds) ICOODB 2009. LNCS, vol. 5936, pp. 77-94. Springer, Heidelberg (2010).
19. Kowalski, Tomasz Marek, Adamus, Radoslaw.: Optimisation of language- integrated queries by query unnesting, Crossref DOI link: https://doi.org/10.1016./J.CL.2016.09.002

