GENERIC APPROACH TO AUTOMATIC INDEX UPDATING IN OODBMS

Tomasz M. Kowalski, Kamil Kuliberda, Jacek Wiślicki and Radosław Adamus
Technical University of Lodz, Stefanowskiego 18/22, 90-924 Lodz, Poland

Keywords: Index maintenance, Automatic index updating, Indexing, Triggers, OODBMS, SBA, SBQL, ODRA.

Abstract: In this paper, we describe a robust approach to the problem of the automatic index updating, i.e. maintaining cohesion between data and indices. Introducing object-oriented notions (classes, inheritance, polymorphism, class methods, etc.) in databases allows defining more complex selection predicates; nevertheless, in order to facilitate selection process through indices, index updating requires substantial revising. Inadequate index maintenance can lead to serious errors in query processing what has been shown on the example of Oracle 11g ORDBMS. The authors work is based on the Stack-Based Architecture (SBA) and has been implemented and tested in the ODRA (Object Database for Rapid Applications development) OODBMS prototype.

1 INTRODUCTION

The general idea of indexing in object-oriented databases does not differ from indexing in relational databases (Elmasri, 2004). Indices are data-structures improving the speed of retrieving database objects according to a given criteria. Database indices ought to ensure transparency, i.e. are used automatically during query evaluation. In this paper authors focus on the index maintenance.

Indices, like all redundant structures, can lose cohesion if the data stored in the database are mutated. Thus, to ensure validity of indices the update of data has to be combined with rebuilding of appropriate indices. The rebuild process should be transparent to relieve a programmer of an inconvenient and error prone task. Furthermore, the additional time required for an index update in response to a data modification should be minimized. This is critical from the point of view of the large databases efficiency. To achieve this, database systems should efficiently find indices which became outdated due to a performed data modification. Next, the appropriate index entries should be corrected so that all index invocations would provide valid answers. However finding a general and optimal solution for index updating is not possible due to the complexity of DBMSs. Such a task requires analysis of many different real life situations occurring in the database environment in order to minimize deterioration of performance.

ODRA OODBMS prototype is based on Stack Based Architecture (SBA) (Adamus, 2008)(SBA&SBQL, 2008). Proposed implementation of indexing in ODRA enables creation, transparent optimisation and automatic maintenance of indices facilitating processing of selection predicates based on arbitrary deterministic expressions.

The rest of the paper is organized as follows. The next section comprises an overview of indexing facilities and theirs limitations with emphasis on capabilities resulting from an approach to the index maintenance, section 3 presents SBA, section 4 describes the approach to index updating in ODRA, section 5 presents a summary and concludes.

2 OVERVIEW OF INDEXING IN RDBMS AND OODBMS

Almost 40 years of research on relational systems resulted in development of various indexing aspects (Elmasri, 2004): *primary index*, *clustered index*, *secondary access paths*, *multi-key index*, development of diverse index structures, *etc*.

In RDBMSs keys defining an index on a table usually are simply values stored in columns. Such index requires straightforward mechanisms

M. Kowalski T., Kuliberda K., Wiślicki J. and Adamus R. (2009).
GENERIC APPROACH TO AUTOMATIC INDEX UPDATING IN OODBMS.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 255-260
DOI: 10.5220/0001834702550260
Copyright © SciTePress

providing user transparency. Modifications to an indexed table are easy to detect by the DB engine Therefore, details of the automatic index updating for RDBMSs are omitted in technical database specifications and considered rather as an implementation issue.

Topics of index organization and optimization in object-oriented database management systems have been deeply researched, e.g. (Bertino, 1997)(Henrich, 1997)(Płodzień, 2000). Major research has been dedicated to cope with improving the efficiency of processing nested query predicates (i.e. accessed using path-expression) and considering inheritance. The most important proposed solutions are Multi-Index, Inherited Multi-Index, Nested-Inherited Index, Path Index, Access Support Relations. In the approach to automatic index updating presented in (Bertino, 1997) all instances of classes associated with a path-expression based index need to be taken into consideration to ensure index validity after data modifications.

The distributed object management system H-PCTE developed in University of Siegen proposed a different solution to automatic index maintenance (Henrich, 1997). The solution handles complex derived attributes defined e.g. employing regular path expressions and aggregate functions. On the other hand, its authors outline some limitations of this approach concerning efficiency and consideration of user-methods giving general suggestions how these disadvantages should be overcame.

Another approach to index maintenance discussed for *function-based indexing* developed in context of Thor, a object-oriented database system by the Massachusetts Institute of Technology uses so-called objects registration schema (Hwang, 1994). Despite the theoretical generality of this approach, it has not been fully implemented.

According to the best of the authors knowledge very little amount of indexing techniques proposed in the scientific literature have been incorporated into commercial OODBMSs products and major prototypes. Careful inspection of applied indexing facilities is possible through analysis of major object-oriented databases.

Transparent indexing in the db4o OODBMSs by db4objects is provided only for attributes of classes defining indexed collections (db4objects, Inc.) The Objectivity/DB by Objectivity additionally supports concatenated index on more attributes (Objectivity, 2006). The only tested database which supports transparent indexing employing path-expressions is GemFire (GemStone, 2008). The ObjectStore supports adding an index defined using complex path-expressions as a key but provides only partial index maintenance transparency. Updating an index entry after data modification must be explicitly triggered by a programmer (Progress, 2008).

The above mentioned prototypes and commercial products provide sufficiently complete overview of the indexing state of art in OODBMSs.

2.1 Limitations of Function-Based Index Maintenance in ORDBMS

The authors did not found any object-relational databases supporting indexing using aggregate functions or path-expressions.

Function-based index (Oracle by Oracle Corporation) (Strohm, 2008) and other similar solutions enabling defining keys using expressions addressing more than one table column and internal functions or user-written functions generally do not introduce conceptual difficulties. Nevertheless, this aspect of indexing becomes complex when object-oriented model and language extensions are considered. The Oracle documentation does not provide extensive information concerning the automatic maintenance of such indices.

To identify properties of the Oracle's approach the authors performed tests introducing a schema shown below:

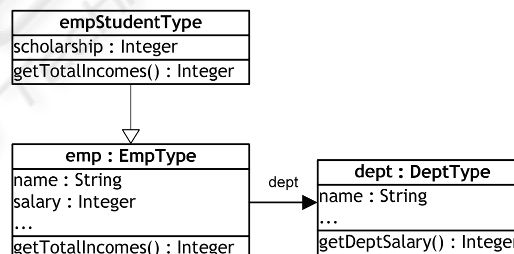

Figure 1: Example object-relational schemata.

The Oracle approach to index updating in case of method-based indices consists in triggering an index update routines during modifications done to any data in a tuple with associated index entries.

The disadvantage mentioned above grows to a large problem in case when the method used to define an index key accesses a data outside an indexed object. For example the method *getDeptSalary* of the *DeptType* has the following definition:

```
CREATE TYPE BODY dept_type IS
 MEMBER FUNCTION getdeptsalary RETURN
   NUMBER DETERMINISTIC IS
 BEGIN DECLARE aux NUMBER;
  BEGIN
```

```
    SELECT sum(salary) INTO aux FROM emp
   e WHERE e.dept.name = self.name;
    RETURN aux;
END; END; END;
```

It accesses not only the given *DeptType* tuple data but also reaches the *emp* collection. This method calculates a sum of department employees salaries and can be used as a selection predicate. Oracle also enables indexing *dept* collection according to *getDepSalary* method:

```
CREATE INDEX dept_getdeptsalary_idx ON
   dept d (d.getDeptSalary());
```

Similarly like in case of *emp_gettotalincomes_idx*, a command altering *dept* tuples triggers updating of the index. However, any modifications done to *emp* objects, e.g.

```
INSERT INTO EMP
   SELECT emp_type ('Smith',350,REF(d))
   FROM DEPT d WHERE d.name = 'HR';
```

are not taken into consideration and *dept_getdeptsalary_idx* index loses cohesion with the data. Unfortunately, queries which use this index, e.g.:

```
SELECT d.name, d.getDeptSalary() FROM
   DEPT d
   WHERE d.getDeptSalary() < 24500;
```

can return incorrect answers, since the selection processes and final results depend on the index content. Hence, the applied index updating solution is not proper to handle indices with keys based on “too complex” methods. In practice *function-based indices* feature in Oracle can lead to erroneous work of database queries and applications.

3 SBA AND SBQL

SBA (Stack Based Architecture) (SBA&SBQL, 2008)(Subieta, 2008) a theoretical and methodological framework for the designed solution presented in this paper. SBA reconstructs query languages’ concepts from the point of view of programming languages (PLs). In SBA a query language (called SBQL – Stack Based Query Language) is considered a special kind of a programming language.

The inherent property of languages based on SBA is the naming-scoping-binding principle. Each name occurring in a query is bound to the appropriate run-time entity (an object, an attribute, a method a parameter, etc.) according to the scope of its name. The principle is supported by means of the environment stack (ENVS) containing *binders*. This mechanism is well known from popular programming languages implementations.

The use of the ENVS can also be extended beyond the query execution to other aspects like runtime support for index updating. This expansion will be presented, as a part of a whole proposed index updating mechanism, in the following section.

4 SOLUTION TO INDEX UPDATING

4.1 Example Database

The schema in Figure 2 is introduced to exemplify indexing description in ODRA OODBMS

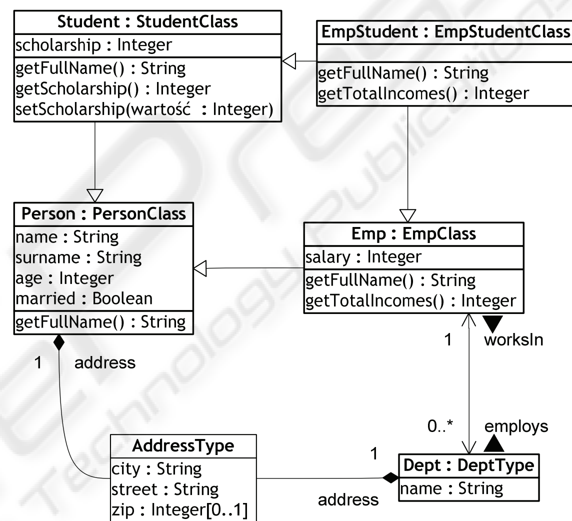

Figure 2: Example object-oriented schema.

The example schema illustrates personnel records of the company. Persistent instances of the depicted classes can be accessed using theirs instance names *Person*, *Student*, *Emp* and finally *EmpStudent*. Using *Person* name in SBQL query results in returning all instances of *PersonClass* class and its subclasses.

Classes can contain methods taking advantage of the polymorphism; hence, are overridden in derived subclasses. E.g. *getTotalIncomes*() method of *EmpClass* returns the value of a *salary* attribute, but for instances of the *EmpStudentClass* it returns sum of *salary* and *scholarship* attributes.

Referring to the data schema in Figure 2 above we introduce the example store shown in Figure 3 presenting classes and objects, their values, identifiers and the most important relations between them.

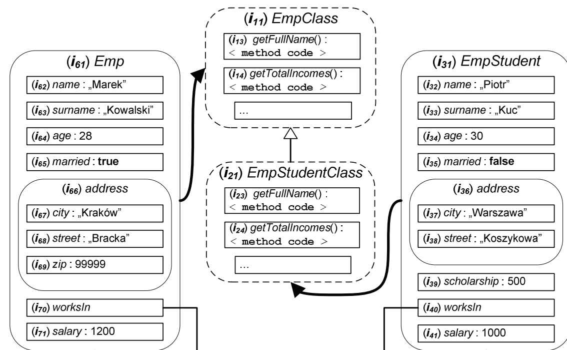

Figure 3: Sample store comprising of classes and objects.

4.2 Indexing in ODRA OODBMS

The theoretical idea for query optimization using indices was developed and presented in (Płodzień, 2000).

The schema in Figure 2 gives the opportunity to present wide variety of indices supported by the ODRA OODBMS indexing engine. The authors approach index updating will be explained using the following example indices:

- *idxAddrStreet* – this is the index which returns address subobjects of *Person* objects according to a *street* attribute. Nonkey objects are defined by a path expression, i.e. *Person.address*.

Nevertheless, even more complex indices can facilitate query processing, e.g.:

- *idxEmpTotalIncomes* – the index which uses an *Emp* class method *getTotalIncomes*() as a key for selecting *Emp* objects. This method is overridden for the *EmpStudent* class instances.
- *idxDeptYearCost* – the index for *Dept* objects based on the **sum**(*employs.Emp.salary*)*12 expression returning an approximate total cost of a given department for the year period.

4.3 Index Update Triggers

Each modification performed on objects (creation, update and deletion) is done by the ODRA object store CRUD (Create, Retrieve, Update, Delete) interface. The proposed approach to index updating concentrates on this element of the system as it is the easiest and certain way to trace data modifications. Possible modifications that can be performed on an object are the following:

- updating a value of an integer, double, string, Boolean, date or object reference,
- deleting,
- adding a child object,
- other database implementation dependent modifications.

The implementation introduces a group of special auxiliary structures called Index Update Triggers (IUT) together with Triggers Definitions (TD). These elements are essential to perform index updating:

- each IUT associates one database object with an appropriate index through a TD. Existing IUTs automatically initialize the index updating mechanism when a modification concerning the given object is about to occur. More than one IUT can be connected with a single object.
- TDs provide means to find objects which should be equipped with IUTs. TD specifies the type of an IUT.

An object is associated with IUTs when it participates in accessing nonkey objects or calculating key values for indices. Therefore, modification to objects not linked with any index does not trigger unnecessary index updating. Altering objects equipped with IUTs is likely to influence topicality of the indices and IUTs.

Our implementation introduces four basic types of IUTs (each IUT refers to different TD type):

- Root Index Update Trigger (R-IUT) – is by default associated with the root database entry which is a superparent for all indexed database objects. When a new object is created in the databases root the trigger can cause generation of an NonkeyPath- or Nonkey- Update Trigger (described below) for the new child object. This trigger is also used to initialize and terminate all triggers associated with an index.
- Key Index Update Trigger (K-IUT) – associated with objects used to evaluate a key value for the specific nonkey object (identifier passed together with TD as an additional parameter). Each modification to such objects can potentially modify the process of evaluating a key and its value. Therefore, an K-IUT is responsible for updating an appropriate index entry and maintaining appropriate K-IUTs associated with the given nonkey object.
- NonkeyPath Index Update Trigger (NP-IUT) – a trigger similar to R-IUT. It is generated when an index nonkey object is defined by a path expression (e.g. *idxAddrStreet* index).
- NonKey Index Update Trigger (NK-IUT) – a trigger that is assigned to indexed (nonkey) objects. It is generated by parent objects update triggers (R-IUT or NP-IUT). The process of creating NK-IUT consists of the following steps:
    - first a *NK-IUT* is assigned to the given nonkey object

- the key value is calculated,
- the corresponding index entry is created (if a valid key value is found),
- K-IUTs are generated and parameterised with the nonkey object identifier.

Basing on the sample store depicted in Figure 3 example IUTs for index *idxAttrStreet* shown in Figure 4 would be generated. Let us assume that i_0 is the identifier of the databases root. Nonkey objects associated with K-IUTs are stated in parentheses.

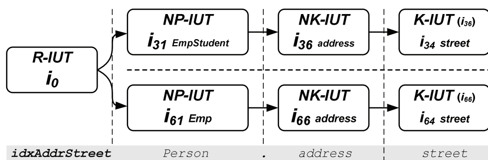

Figure 4: Example of index update triggers.

4.4 The Approach to the Index Update Process

The architectural view of the proposed index update process is presented in Figure 5.

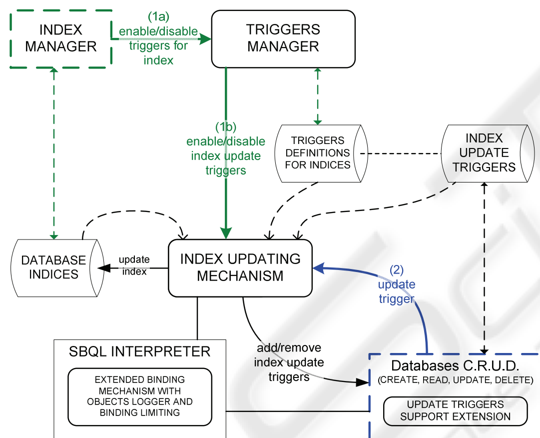

Figure 5: Index Updating Engine architecture.

When the administrator adds an index, TDs are created before IUTs (this step is shown using the green coloured arrows numbered 1a and 1b):

1. Index Manager initializes a new index and issues Triggers Manager a message to build TDs,
2. next, the Triggers Manager activates the Index Updating Mechanism which basing on the knowledge about indices and TDs proceeds to add IUTs.

Removing an index causes removal of IUTs (together with NK-IUTs corresponding index entries are deleted) and TDs. The mediator managing the addition and removal of IUTs is a special extension of the CRUD interface.

The second case when the Index Updating Mechanism is activated is when the store CRUD interface receives a message to modify the object which is marked with one or more IUTs (shown in Figure 5 using the blue coloured arrow with number 2). CRUD notifies the Index Updating Mechanism about forthcoming modifications and all necessary preparation before database alternation are performed. This step is particularly important in case of changes which can affect a key value for the given nonkey object. It consists of:

1. locating the index entry which corresponds to the nonkey object (key value is necessary),
2. identifying objects that are accessed in order to calculate the key value (they are equipped with an identical K-IUT).

After gathering required information CRUD performs requested modifications and following operations are executed by the Index Updating Mechanism:

1. update of index entries for the given nonkey object by,
2. update of existing IUTs by generating new or removing outdated ones.

4.5 SBQL Interpreter and Binding Extension

A significant element used by the Index Updating Mechanism is the SBQL interpreter (also shown in Figure 4) extended with the ability to:

- Log database objects during evaluation of an index key expression. This process occurs during binding object names on ENVS (other database entities like procedures, views, etc. and literals are discarded) – this feature is used to locate all objects which are or should be equipped with *K-IUTs*.
- Limit the first performed binding only to one specified object – this feature significantly accelerates and facilitates verification whether a new child sub-object added to an object with R-IUT or NP-IUT should be equipped with NP-IUT or NK-IUT, i.e. to check whether a new child is a nonkey object or potential superparent of a nonkey object.

The basic functions of the interpreter used by the Index Updating Mechanism are:

- traversing from the databases root or objects equipped with NP-IUTs to nonkey objects,
- generating a key value for nonkey objects.

5 CONCLUSIONS

The ODRA OODBMS is a proof of concept prototype as well as the implemented *Index Updating Mechanism*. The fair comparison can be conducted considering general properties to the index maintenance and its influence on capabilities of a database indexing.

An undoubtful advantage of the index updating approach in majority of relational and object-relational databases is an economic usage of the data store. In the implemented solution for the ODRA OODBMS index update triggers are in many cases written together with complex objects and objects containing single values. This situation is acceptable considering that nowadays databases administer a very large amount of memory (or disk) space.

Improving performance among other depends on the diversity of used index structures and flexibility in defining an index. The properties of the SBQL language, i.e. orthogonality and compositionality, enable easy formulating of complex selection predicates (including usage of complex expressions with polymorphic methods and aggregate operators). The proposed organization of indexing in ODRA OODBMS provides all necessary mechanisms for creating indices with keys based on such predicates. As it was shown, the most popular DBMSs do not provide similar indexing capabilities and flexibility. Furthermore, attempts to extend the power of indexing facilities (Oracle *function-based indices*) lead to mistakes in obtaining queries results and erroneous work of database applications.

In contrast to all implemented solutions to the automatic index updating issue presented in research literature or incorporated in commercial products, the authors approach based on *Index Update Triggers* implemented in ODRA OODBMS provides transparent, complete and generic support for variety of index definitions. Moreover, the additional data modification costs associated with the index maintenance concerns exclusively objects used to access the indexed objects or to determine a key value. One can argue about increased storage cost caused by *IUTs*. Nevertheless, as it is shown in (Bertino, 1997) the maintenance of indices defined using complex expressions require introducing a lot of additional information in the index structure. Other advantage of the authors *IUTs* set on objects used to determine a key value is that they include direct reference to an indexed object, whereas other solutions are often forced to identify it indirectly (e.g. by reverse navigation methods) (Bertino, 1997)(Henrich, 1997).

The presented solution to index updating in OODBMS is generic and versatile; however, it requires optimizations to avoid unnecessary performance deterioration particularly in simple updating cases. Such optimizations have been developed and partially implemented. Discussion concerning this subject is omitted due to the paper size restrictions.

ACKNOWLEDGEMENTS

This research work is funded from the science finances in years 2008/2009 as a research project nr N516 383334.
R. Adamus and J. Wislicki are scholarship holders of project entitled "Innovative education..." supported by European Social Fund.

REFERENCES

Adamus R., Kowalski T.M., Subieta K., et al., 2008. *Overview of the Project ODRA*. Proceedings of the First International Conference on Object Databases, ICOODB 2008, Berlin, ISBN 078-7399-412-9, pp. 179-197.

Bertino E. et al., 1997. *Indexing Techniques for Advanced Database Systems*. Kluwer Academic Publishers, Boston Dordrecht London.

db4objects Inc., 2008. *db4o Tutorial for Java*. Production Release V6.3.

Elmasri R. and Navathe S. B., 2004. *Fundamentals of Database Systems 4th ed.* Pearson Education, Inc., ISBN: 83-7361-716-7

GemStone, 2008. *GemFire Enterprise Developer's Guide*, Version 5.7.

Henrich A., 1997. *The Update of Index Structures in Object-Oriented DBMS.* Proceedings of the Sixth International Conference on Information and Knowledge Management (CIKM'97), Las Vegas, ACM 1997, ISBN 0-89791-970-X: pp. 136-143.

Hwang *D. J.*, 1994. *Function-based indexing for object-oriented databases*. PhD thesis, Massachusetts Institute of Technology.

Objectivity, 2006. *Objectivity/SQL++*. Part Number: 93-SQLPP-0, Release 9.3.

Płodzień J., 2000. *Optimization Methods In Object Query Languages*. PhD Thesis. IPIPAN, Warszawa.

Progress Software Corporation, 2008. *ObjectStore Java API User Guide.* ObjectStore, Release 7.1 for all platforms.

SBA & SBQL Web pages: http://www.sbql.pl/

Strohm R., et al., 2008. *Oracle® Database Concepts*. 11g Release 1 (11.1), Part Number B28318-05.