Composite-Object Views in Relational DBMS: An Implementation Perspective

(Extended Abstract[1])

H. Pirahesh, B. Mitschang[2], N. Südkamp[3], B. Lindsay
IBM Almaden Research Center
San Jose, CA 95120, USA
e-mail:{pirahesh, bruce}@almaden.ibm.com

Abstract. We present a novel approach for supporting Composite Objects (CO) as an abstraction over the relational data. This approach brings the advanced CO model to existing relational databases and applications, without requiring an expensive migration to other DBMSs which support CO. The concept of views in relational DBMSs (RDBMS) gives the basis for providing the CO abstraction. This model is strictly an extension to the relational model, and it is fully upward compatible with it. We present an overview of the data model. We put emphasis in this paper on showing how we have made the extensions to the architecture and implementation of an RDBMS (Starburst) to support this model. We show that such a major extension to the data model is in fact quite attractive both in terms of implementation cost and query performance. We introduce a CO cache for efficient navigation through components of a CO. Our work on CO enables existing RDBMSs to incorporate efficient CO facilities at a low cost and at a high degree of application reusability and database sharability.

1 Motivation

It is widely agreed now that complex applications, such as design applications, multi-media and AI applications, and even advanced business applications can benefit significantly from database interfaces that support *composite (or complex) objects* (shortly, CO). A generally accepted characterization defines a CO consisting of several components (possibly from different types) with relationships in between [2,3,17,11]. Interestingly, object oriented DBMSs (OODBMSs) have adopted a very similar model [1,21,9,13]. This is particularly true for OODBMSs used in practice. Especially OO programming environments have made advances in handling of COs. Such environments facilitate the growth of complex applications. As a result, there is considerable pressure on RDBMSs for better support of COs. To respond to this demand, several systems today are bridging OO environments with relational. An example of such a system is the Persistence DBMS [12], which builds a layer on top of RDBMSs, providing better support for COs. These systems essentially extract data from a relational database and load it into OO environments.

In summary, relational systems to be viable must be able to understand COs (a language extension issue), and must be able to handle them well (optimization issue). Given the prevalence of SQL as a database interface language both for application and database interoperability, we have proposed an extension to SQL to handle COs. In our approach, called SQL Extended Normal Form (for short XNF) and introduced in [18], we derive COs from relational data. That paper covers the language part giving details on syntax and semantics. One major achievement is that such a powerful extension was made with full upward compatibility with SQL, hence opening up a path for the current DBMSs in research

1. The EDBT committee decided to present the full version of the paper in Information Systems, special issue: 'extended database technology', Spring 1994.

2. University of Kaiserslautern, Dept. of Computer Science, P.O.Box 3049, 67653 Kaiserslautern, Federal Republic of Germany, e-mail: mitsch@informatik.uni-kl.de

3. IBM Heidelberg Scientific Center, Vangerowstr. 18, 69115 Heidelberg, Federal Republic of Germany, e-mail: suedkamp@dhdibm1.bitnet

and industry to move forward toward handling of COs. In this paper, we attack the implementation issues, showing how to evolve a relational DBMS to handle this problem. Again, we show a growth path for RDBMSs. As mentioned before, good optimization is essential. Given the extensive optimization techniques implemented in RDBMSs, we have paid attention to reusing these capabilities. From a practical viewpoint, this is very attractive since the optimization component is not cheap.

2 XNF Language Overview: Basic Concepts, Syntax, and Semantics

XNF's notion of CO is based on the view paradigm. Instead of a single normalized table, as in standard relational systems an XNF query derives an ER-like structure [4,21,11,13] from an underlying relational database. Those *CO views* (also called object views [16,22] or structured views) are defined using a powerful *CO constructor* and consist of component tables and explicit relationships between components.

An XNF query is identified by the keywords OUT OF and consists of the following parts:

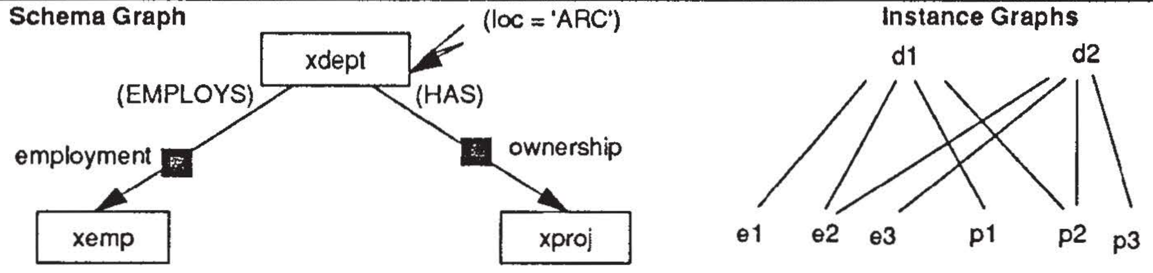
- definitions for the component tables, in general identified by the keyword SELECT,
- definitions for the relationships, identified by the keyword RELATE, and
- specifications for the output, identified by the keyword TAKE.

Table and relationship definitions are mainly expressed using existing SQL language constructs. They make out XNF's CO constructor which can be seen as a proper extension to SQL by a compound query statement. With this, an XNF query simply reads like this:

'**OUT OF** ... *the CO* (that is constructed by the CO constructor

TAKE ... *the parts projected* (that define the resulting CO)'

As an introduction to XNF syntax and semantics, let us discuss the example CO abstraction called *dep_ARC* given in Fig.1. The upper part of Fig.1 shows on the left the schema and on the right the instance level showing the COs derived, whereas the lower part of Fig.1 gives the corresponding CO query that defines this abstraction.



```
XNF Query  CREATE VIEW dep_ARC
           AS
           OUT OF xdept   AS   (SELECT * FROM DEPT WHERE loc = 'ARC'),   definition of XNF
                  xemp    AS   EMP,                                      component tables
                  xproj   AS   PROJ,
                  employment AS  RELATE  xdept VIA EMPLOYS, xemp
                                 WHERE   xdept.dno = xemp.edno),         definition of XNF
                  ownership  AS  RELATE  xdept VIA HAS, xproj            relationship tables
                                 WHERE   xdept.dno = xproj.pdno),
           TAKE *          /* projection of all XNF component and relationship tables */
```

Figure 1: Sample CO 'dep_ARC'

This XNF query retrieves the departments located at 'ARC', and to each one the corresponding employees as well as the projects are connected. As shown by this sample XNF query, the nodes, i.e. the component tables, are derived through standard SQL queries. Syntactic short-cuts (see definition of *xemp* and *xproj* component tables) are provided for sake

of brevity. In our example the base tables departments (*DEPT*), employees (*EMP*) and projects (*PROJ*) of the underlying relational database are used for derivation. The relationship tables that make up the edges of the query's schema graph show a different syntax, but basically also adopt SQL query facilities. The RELATE clause gives the parent table first and then the child table, and the WHERE clause holds the (standard SQL join) *predicate* that specifies the criteria for relating the partner tuples via *connections*. In order to read the relationship-defining query expressions in a convenient way, we have given role names (VIA clause) to the parent partners of the relationships. Based upon their predicates, the relationships establish for any given department connections to the employees it *EMPLOYS* and to the projects it *HAS*.
Retrieval of such an XNF CO results in retrieval of all the tuples of the component tables and provision for the relationship information, i.e. connections defined by the XNF relationships. So far, an XNF CO specifies a heterogeneous set of records with different record formats. If a component tuple is used multiple times within a view, then it exists, of course, only once in the view, but it participates in multiple connections (possibly from different relationships). Therefore the important notion of *object sharing* (illustrated by the instance graphs in Fig.1 showing the employees *e2* and *e3* as shared objects) is a fundamental part of the XNF CO concept. An XNF query may also specify a *recursive CO* being identified by a cycle in the query's schema graph. This cycle basically defines a 'derivation rule' that iterates along the cycle's relationships to collect the tuples until a fixed point is reached and no more tuples qualify.
XNF COs may be combined, projected, and restricted. Combination is done by simply defining a relationship between any node of one CO and any node of another one. Projection is defined by listing all the nodes and relationships to be retained. The star '*' is used as a special syntactic construct for projection of all the components with their attributes and all the relationships defined. Restriction can be done through additional predicates on the node tables and the relationships. There is also a set of CO update operators, enhancing the interface to handle insert, read, update, and delete operations. In addition, the interface supports connect and disconnect operations on relationships.
All retrieval and manipulation operations of the XNF language work at the XNF level, taking into account the given graph structure and the heterogeneous tuple set. Since the result of an XNF query consists of a set of component tables and relationships, an XNF query (or XNF view) can be used as input for a subsequent XNF query or view definition. This is also true for all other XNF operations. Therefore the model is closed under its language operations. More information on the XNF language, the multi-lingual API, and (update) semantics can be found in [18].

3 Composite Object Processing

Since both the XNF language as well as the XNF API are built on SQL ideas, we decided to develop the XNF system as an *extension* to an existing RDBMS rather than building a new DBMS. Hence we advocate for an integrated DBMS, which handles both the tabular as well as the CO data. In doing so, we are able to reuse important system features that have taken years to build and are vital for e.g. system robustness, failure tolerance, and system performance. Especially, since the specification of XNF views mostly reuses the relational query language (SQL in our case), almost all of the optimization techniques developed in the context of RDBMSs remain applicable. From a technical viewpoint (and also from an economic one) we chose Starburst DBMS [10] as the starting point. Starburst was particularly attractive due to its extensibility features as we will see shortly.

3.1 Query Processing Architecture

The query processing architecture of Starburst incorporates the *query language processor* CORONA [10] and the *data manager* CORE [14]. CORONA compiles queries (written in extended SQL) into calls to the underlying CORE services to fetch and modify data. As depicted in Fig. 2, there are five distinct stages of query processing in CORONA; each stage is represented by a corresponding system component. For the moment, we consider only the unshaded parts of Fig.2; the shaded areas mark XNF extensions and will be discussed in the next subsection.

An incoming SQL query is first broken into tokens and then parsed into an internal query representation called *query graph model* (shortly QGM). Only valid queries are accepted, because *semantic analysis* is also done in this first stage. During *query rewrite*, the QGM representation of the query is transformed (rewritten by transformation rules) into an equivalent one that (hopefully) leads to a better performing execution strategy when processed by the subsequent stage of plan optimization. *Plan optimization* chooses a possible execution strategy based on estimated execution costs, and writes the resulting *query execution plan* (QEP) as the output of the compilation phase. This evaluation plan is then repackaged during the *plan refinement* stage for more efficient execution by the *query evaluation system* (QES). At runtime QES executes the QEP against the database.

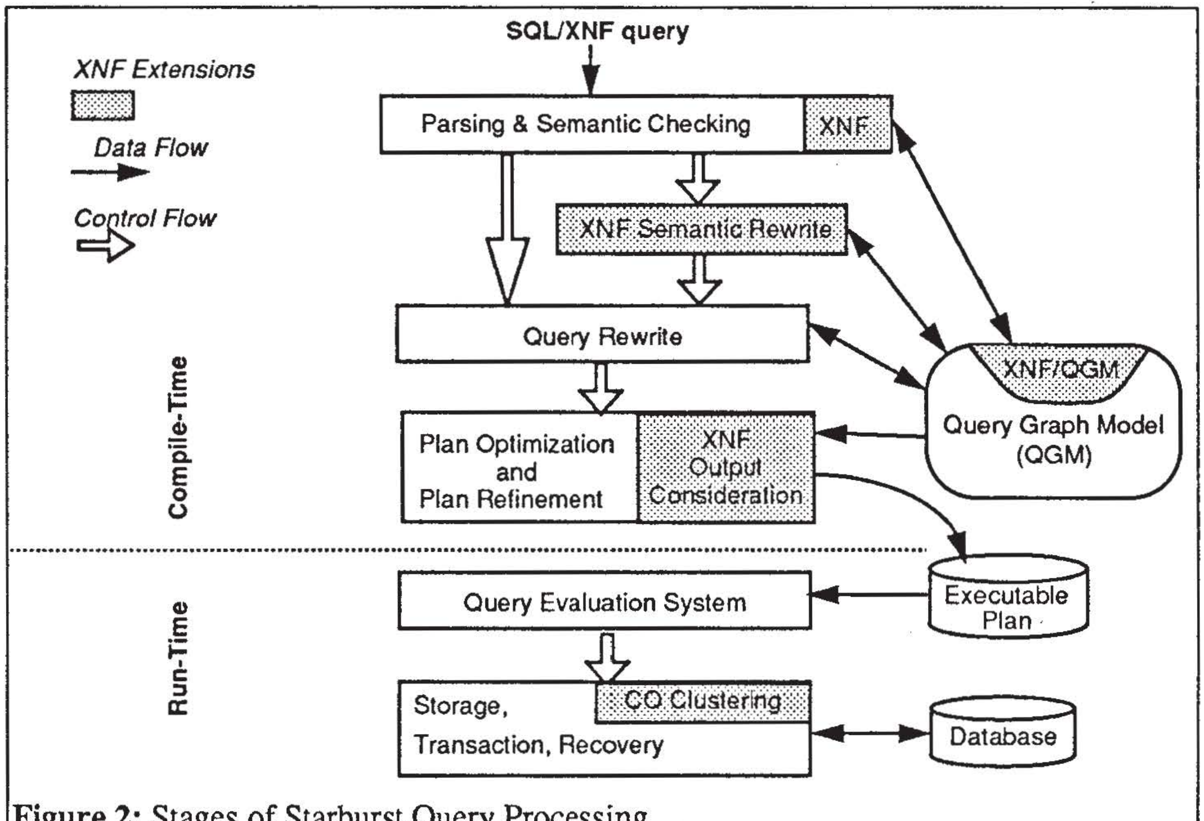


Figure 2: Stages of Starburst Query Processing

3.2 CO Processing Steps

The distinguished stages of XNF's CO query processing are shown in Fig.2. Those features that are different to the ones used in the traditional Starburst are shaded. Fig.2 already exposes that the XNF language processor is truly an extension to the SQL processor. XNF queries are translated to a form very close to standard SQL, allowing reuse of the extensive optimization and evaluation machinery of the RDBMS with little change. This translation is performed at compile time, and is optimized, eliminating any runtime overhead.

3.3 XNF Semantic Checking

The crucial extension to the relational case was the CO constructor. Since this extension affected the language grammar, both the language parser and the semantic checking had to be extended correspondingly. In the same way as the standard SQL processor created during this phase the internal query representation, i.e., a normal form QGM graph (for short NF QGM), the XNF processor had to create the so-called XNF QGM graph that has to incorporate the XNF query semantics. In order to do this, a new operator had to be installed for QGM. The purpose of this *XNF operator* is to reflect the semantics of the language's CO constructor. Therefore, the XNF operator had to be able to incorporate n>=1 incoming tables and to produce m>=1 output tables being the resulting node tables and relationship tables of the CO constructed. In addition to this, regular output processing had to be modified to allow generation of a heterogeneous set of tuples in the answer set (generation of tuples belonging to different nodes and relationships). This is done by the so-called 'top' operator, which deals with the interface between the query processor and the application program. Each QGM graph has a single top operator.

In the first stage of XNF query compilation the internal query representation is built by means of the XNF semantic routines. As already mentioned, XNF QGM uses the XNF operator in order to incorporate XNF query semantics.

Since an XNF query consists of three building blocks, there are also three semantic routines that together construct the final XNF query graph:

(0) QGM initialization
(1) Derivation of XNF component tables
(2) Consideration of component restrictions and XNF predicates
(3) Handling projection.

3.3.1 XNF Semantic Rewrite

In this step the translation from XNF QGM (and XNF semantics) to NF QGM (and NF semantics) has to be accomplished, thereby transforming an XNF query graph into a semantically equivalent NF query graph. Speaking in other words, this component has to replace the XNF operator and the XNF predicates by corresponding NF operators organized in an NF QGM graph. In this step we exploit that the components (i.e., the building blocks) of COs are tables, whose derivation is already specified via NF query graphs within an XNF operator (see Sect. 4.1). XNF semantic rewrite proceeds in two major steps:

(1) Removal of the XNF operator box
(2) Consideration of XNF predicates, e.g. reachability.

Finally, there is only one NF QGM graph constructed for such a multi-table XNF query, i.e. common subexpressions are immediately installed such as the derivation of a component is also used for the derivation of its child(ren) component(s), and both are used for the relationship derivation. Comparing this multi-table derivation as applied by XNF with the single component derivation in SQL clearly shows the impact of XNF's inherent treatment of common subexpressions and that the XNF rewrite approach is optimal w.r.t. processing common subexpression. A formal proof of this is beyond the scope of this paper.

3.3.2 Rewrite and Plan Optimization

Since the previous step already produced a clean NF QGM (that reflects the CO query semantics), the remaining compilation work can be done by the components of the original SQL language processor. That is, the NF QGM graph built by XNF semantic rewrite is transformed by the NF query rewrite component to a semantically equivalent one that, in general, allows more efficient evaluation strategies to be chosen for the QEP when being processed by the plan optimization and query refinement components. For example, existential subqueries can be converted into joins. Remember, all these components are shared between the XNF language processor and the SQL language processor. A detailed description of these components can be found in [10,19].

4 Data Extraction, XNF Cache, and API

In addition to standard SQL cursor support, we allow the retrieval of *all tuples* contributing to the result of an XNF query and the materialization of the *complete* COs. The XNF processing model has been designed for a workstation/server environment, where the database server can deliver complete COs on request.

In Fig.3 we show the overall structure of our prototype implementation. An application sends a data request, i.e. an XNF query to the DBMS. Query translation and optimization takes place as described in the previous Section (these compile-time activities are marked by the shaded area in Fig.3). At runtime the generated query plan is executed and the complete result is delivered by the database system, converted into an internal main-memory representation and made accessible to the application program via the cursor interface. For long transactions, XNF allows the cache to be stored on disk and retrieved later, thereby protecting the cache from client machine's failure.

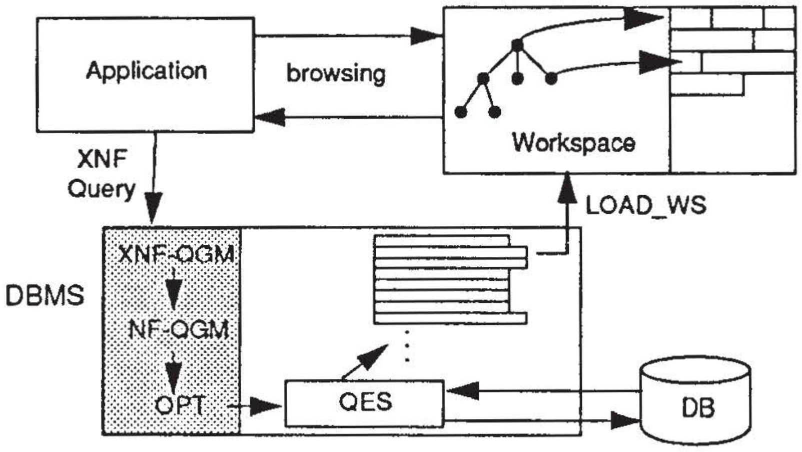


Figure 3: XNF Cache, Data Extraction, and API

The workspace is constructed from the output tuples of the XNF query by converting connections into pointers which allow traversing the structure in any direction. In addition we generate pointers to allow browsing all elements of a component and all elements of a node which are connected to a given component by a specified relationship. These pointers provide primitive access support for the cursors to be defined upon the component tables.

In our prototype we have implemented a subset of XNF API and the XNF cache manager. We have used this prototype to measure the performance of XNF. One significant result is that the performance of XNF cache is quite comparable with fast OODBMSs reported in Cattell's benchmark [8]. Using the traversal operation from that benchmark, we could access in a pre-loaded XNF cache more than 100,000 tuples per second which matches the requirements for CAD applications.

5 Conclusions

The novel approach of supporting COs as an abstraction over relational data is quite attractive. This approach brings the advanced CO model to existing relational databases and applications, without requiring an expensive migration to other DBMSs which support COs. The salient features of the approach are:

- a data model and query language that unifies CO and relational constructs by means of CO views,
- an elegant implementation approach that guarantees efficient data extraction, and
- a multi-lingual API with efficient navigation and manipulation facilities, and a seamless C++ interface to the cached data.

XNF defines an evolution path from RDBMSs to Composite Object DBMSs. It is interesting that we needed to make very little changes in order to make RDBMSs capable of efficiently handling COs. This is a tremendous advantage, and is very significant in practice. Therefore we called XNF an *enabling technology*. It enables RDBMSs to be extended to deal with CO and CO processing patterns. The technology is inexpensive because it heavily reuses already existing query processing components (with only comparatively small changes), and other system components as for example transaction, recovery, and storage management are totally kept unchanged. Although XNF technology largely builds upon basic relational technology, further extensions (e.g. parallelism and clustering facilities) introduced to the relational part of the system become automatically available to XNF.

XNF technology has been successfully integrated into and is now operational in the Starburst extensible database system developed at IBM Almaden Research Center. Although the extensibility feature provided by Starburst helped a lot in integrating the XNF technology, there is at least conceptually no problem in getting it also into other (non-extensible) DBMS. This is due to the fact that XNF clearly extends SQL, and XNF technology rewrites a CO query into a semantically equivalent NF query. Therefore, only the rewrite component as well as the language extensions have to be incorporated into the query processing component of a DBMS. Extensibility just simplifies that attempt.

Another major conclusion drawn from the discussion so far, should perceive XNF as a high performance approach - as an enabling technology - that provides a path for incorporating relational data into any CO application similar to the Persistence DBMS [12] already mentioned. For example, we can use an XNF DBMS (e.g., the Starburst DBMS presented here) to provide server services to an object-oriented programming system running on the application site. This idea was realized in the prototype system called 'Object/SQL Gateway' [15] that provides object-oriented access to data residing in a relational DBMS. This gateway connects the object-oriented DBMS ObjectStore [13] to the Starburst relational DBMS exploiting XNF technology. It is a first step in providing an integrated access to both types of DBMS using a uniform object-oriented interface. Here, XNF's multi-query optimization helps in considerably reducing the cost of data extraction from relational repository into an object cache. Another important issue is improving XNF query processing with special emphasis on CO cluster facilities, as well as on parallelism technology [6,7,20]. Further, in trying to assess XNF technology, there is already considerable confidence that the query processing concepts for COs presented (especially the multi-query framework as well as its inherent exploitation of common subexpressions) plays an integral part also in query processing for object-oriented languages as well as for deductive database languages [5].

Acknowledgments

The cooperation of the whole Starburst staff is gratefully acknowledged. Special thanks are due to Peter Pistor, who helped in our joint effort of getting the good stuff into XNF, whilst streamlining the syntax and semantics. Guy Lohman improved the optimizer to handle our complex queries, and George Wilson provided valuable implementation experiences in his work on an earlier prototype. We would like to acknowledge V. Srinivasan and T. Lee of DBTI for their work on XNF cache manager and C++ application interface. We would also like to thank J. Thomas for his comments on an earlier version of this paper.

References

1. Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S.: The Object-Oriented Database System Manifesto, in: Proc. of the 1st Int. Conf. on Deductive and Object-oriented Databases, Kyoto-Japan, Dec. 1989, pp. 40-57
2. Albano, A., Ghelli, G., Orsini, R.: A Relationship Mechanism for a Strongly Typed Object-Oriented Database Programming Language, in: Proc. 17th VLDB Conf., Barcelona, 1991, pp. 565-575
3 . Batory, D.S., Buchmann, A.P.: Molecular Objects, Abstract Data Types, and Data Models, in: Proc. 10th VLDB Conf., Singapore, 1984, pp. 172-184
4. Chen, PP: The Entity Relationship Model: Toward a Unified View of Data, in: ACM Trans. on Database Syst., Vol.1, No.1, 1976, pp. 9-36
5. Cheiney, J., Lanzelotte, R.: A Model for Optimizing Deductive and Object-Oriented DB Requests, in: Proc. of Data Engineering Conf., Phoenix, February, 1992
6. DeWitt, D., Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems, in: CACM, Vol. 35, No. 6, 1992, pp. 85-98
7. Graefe, G.: Volcano, an Extensible and Parallel Query Evaluation System, Research Report University of Colorado at Boulder, CU-CS-481-90, 1990
8. Gray, J. (ed.): The Benchmark Handbook for Database and Transaction Processing Systems, Morgan Kaufman Publ. Inc. (1991)
9. Guzenda, L, Wade: ANS OODBTG Workshop position paper, Objectivity, Inc., in Proc of the First OODB Standardization Workshop, May 22, 1990
10.Haas, L., Freytag, J.C., Lohman, G., Pirahesh. H.: Extensible Query Processing in Starburst, in: Proc. of the ACM SIGMOD Conf., Portland, 1989, pp. 377 - 388
11.Kim, W.: Introduction to Object-Oriented Databases, MIT Press, (1991)
12.Keller, A., Jensen R., Agrawal, S.: Persistence Software: Bridging Object-Oriented Programming and Relational Database, in: ACM SIGMOD Conf., 1993, pp. 523-528
13.Lamb, C., Landis, G., Orenstein, J., Weinreb, D.: The Objectstore Database System, in: Communications of the ACM, Vol. 34, No. 10, 1991, pp. 50-63
14.Lindsay, B., McPherson, J., Pirahesh, H.: A Data Management Extension Architecture, in: Proc. of the ACM SIGMOD Conf., San Francisco, 1987, pp. 220-226
15.Lee, T., Srinivasan, V., Cheng, J., Pirahesh, H.: Object/SQL Gateway, presented at OOPSLA workshop, 1993
16.Lee, B.S., Wiederhold, G.: Outer Joins and Filters for Instantiating Objects from Relational Databases through Views CIFE Technical Report, Stanford Univ., May 1990
17.Mitschang, B.: Extending the Relational Algebra to Capture Complex Objects, in: Proc. 15th VLDB Conf., Amsterdam, 1989, pp. 297-305
18.Mitschang, B., Pirahesh, H., Pistor, P., Lindsay, B., Südkamp, N.: SQL/XNF - Processing Composite Objects as Abstractions over Relational Data, in: Proc. of Ninth Int. Conf. on Data Engineering, April 1993, Vienna, pp. 272-282
19.Pirahesh, H., Hellerstein, J., Hasan, W.: Extensible/Rule Based Query Rewrite Optimization in Starburst, in: Proc. of the ACM SIGMOD Conf, San Diego,1992, pp.39-48
20.Pirahesh, H., Mohan, C., Cheng, J., Liu, TS, Selinger, P.: Parallelism in Relational Data Base Systems: Architectural Issues and Design Approaches, in: Proc. of the Int. Symposium on Databases in Parallel and Distributed Systems, Dublin, 1990
21.Zdonik, S., Maier, D: Fundamentals of Object Oriented Databases. Readings in Object-Oriented Database Systems, ISBN 1-55860-000-0, ISSN 1046-1698, Morgan Kaufmann Publishers, Inc., (1990)
22.Zdonik, S.: Incremental Database Systems, in: Proc. of the ACM SIGMOD Conf., Washington, 1993, pp. 408-417