in: Proc. Dagstuhl-Workshop, 1992

Optimization of Complex-Object Queries in PRIMA - Statement of Problems

Harald Schöning
University of Kaiserslautern
P.O.-Box 3049
6750 Kaiserslautern
Federal Republic of Germany
email: schoenin@informatik.uni-kl.de

Abstract

The molecule-atom data model allows the dynamic construction of complex objects using an identifier-reference concept. The model and its implementation in the PRIMA system are sketched. Then, with the help of some sample queries, some alternatives for query evaluation are discussed. The decision among the possible algorithms cannot be based on the standard statistics such as distribution of an attribute value. The kind of information needed is the correlation between graph properties and attribute values. In order to accelerate access, its components may be read in parallel. Using a maximum degree of parallelism for this purpose has disadvantages if the complex object does not qualify with respect to the condition included in the query. Criteria for an optimal degree of parallelism still have to be found.

1. Introduction

The support of complex objects has become a strong requirement for advanced database management systems. There have been several proposals for so-called structurally object-oriented database management systems which focus on complex object handling facilities. In addition, the object-oriented database systems also cover several aspects of complex objects management. In this paper we will consider a specific structurally object-oriented database management system called PRIMA, which provides an elaborated complex-object data model with a variety of underlying storage structures. The data model together with the storage structures and the parallelism offered by the system architecture give several degrees of freedom for the choice of query evaluation algorithms. We focus on a basic operator in PRIMA and demonstrate the possible evaluation methods. In order to determine the optimal combination of choices one has compare the costs of several alternatives which heavily depend on the graph structure of the underlying database. Statistically capturing this graph structure seems to be a hard problem.

The rest of the paper is organized as follows. Section 2 gives a short overview of the molecule-atom data model provided by PRIMA. The architecture of PRIMA is sketched in section 3. Section 4 gives an overview of the query processing in PRIMA. Section 5 focuses on the basic operator in molecule

construction and discusses the choices to be made during optimization. Section 6 gives a short conclusion.

2. Basic Features of the Molecule-Atom Data Model

The molecule-atom data model (MAD model [Mi88a]) has been introduced to support the use of dynamically defined complex objects. Complex object types (molecule types) are defined in terms of their components, which may be either complex object types or basic object types (atom types). An atom type consists of some attribute types, and therefore may be compared to a relation in the relational model. The corresponding objects, called atoms, are similar to tuples. We allow a richer selection of data types than most conventional database systems do. Particularly, the two special data types IDENTIFIER and REFERENCE are used to explicitly express binary relationships between atoms. The *IDENTIFIER* attribute, which is present in each atom exactly once, contains a system-defined primary key (surrogate) to uniquely identify the atom. *REFERENCE* attributes contain one or more IDENTIFIER values, all pointing to atoms of the same atom type (typed references). At the schema level, there must be a corresponding "back reference" for each REFERENCE attribute, i.e., if atom type *A* has a REFERENCE attribute pointing to atom type *B* (for short, "REFERENCE attribute to *B*"), then atom type *B* must contain a corresponding REFERENCE attribute to *A*. Symmetry is also required at the instance level. If the REFERENCE attribute of atom *a* contains the value of the IDENTIFIER attribute of atom *b* the corresponding REFERENCE attribute of atom *b* has to contain the value of the IDENTIFIER attribute of *a*. This structural integrity is enforced by the operations of the MAD model. Thus, there is a means to reflect 1:1, 1:n, and n:m relationships among atoms in a direct and symmetric way. The relationships between atoms, which are manifested by the values of the REFERENCE attributes, lead to the so-called atom network. As described so far, the MAD model is similar to the entity-relationship model [Ch76].

The relationships installed by REFERENCE attributes can be used to define molecule types. The notation `A.bref-B.cref-C` means, for example, that for each atom *a* of type *A* all atoms of type *B* ("*B* atoms") that are referenced by *a*'s REFERENCE attribute *bref* and all *C* atoms referenced by attribute *cref* of these *B* atoms are grouped to a molecule. This definition assigns a direction to the relationships between *A* and *B*, and *B* and *C* respectively. We call such a directed relationships within one molecule a *link*. Hence, a molecule can be seen as a directed subgraph of the atom network having one root, the so-called *root atom* (in contrast to the *component atoms*). If *A* has only one REFERENCE attribute to *B*, `A-B` may be written instead of `A.bref-B`.

Besides hierarchical structures as introduced above, the MAD model also allows network-like and recursive molecule type definitions. When an atom type has more than one predecessor in the molecule type graph, an atom of this type belongs to a corresponding molecule only if it has references to at least one atom of each of the predecessor types that also belongs to that corresponding molecule (AND semantics). Recursive molecule types repeat a component molecule type at several re-

cursion levels. Recursion level 0 of a recursive molecule consists of a molecule of the component molecule type. All molecules of the component molecule type that are referenced by the recursion-defining REFERENCE attribute of a component molecule at level i form level i+1 of the recursion. If a component molecule appears at more than one level of recursion, it belongs to only the lowest numbered one. Thus, cycles in the recursive molecule are avoided, and the termination of the operator computing the transitive closure is guaranteed. Figure 1 illustrates some molecule types and corresponding molecules for a sample database. Several features should be observed:

- Molecules vary in size. For instance, one of the molecules of type A-B-C in Figure 1 consists of only one atom. This *structural incompleteness* is caused by the fact that a REFERENCE attribute has an empty value. Cardinality restrictions may be specified in the database schema which prohibit empty or undefined values for REFERENCE attributes.

- The links from an atom to its descendants may vary from molecule to molecule (consider atom *b1* in molecule type A-(B,D)-C). The molecule-atom data model directly reflects sharing of components among molecules (i.e., molecules overlap[1]). Furthermore, if there are several paths to an atom in a molecule graph, this atom is not replicated, but included only once in the molecule.

Sets of molecules can be inserted, deleted, updated or retrieved (selected) using the SQL-like **m**olecule **q**uery **l**anguage (MQL). In this paper, we concentrate on the SELECT-statement, which is the most complex statement. It is used to extract a set of molecules of a certain type from the database. Its general form is

SELECT <projection clause>
FROM <molecule type definition clause>
WHERE <restriction clause>

The **molecule type definition clause** determines the environment (molecule type) to work on. Besides the facilites for molecule type definition which were introduced so far, a Cartesian Product of molecule types may be defined by enumerating the participating molecule types separated by a comma.

The **restriction clause** contains a condition ranging over the environment molecule type. It may be of arbitrary complexity, and may contain quantifiers and SELECT-statements (nested subqueries). Only those molecules that fulfill this condition (molecules that *qualify*) are members of the result set. If no restrictions are to be imposed, "WHERE <restriction clause>" may be omitted.

1. To emphasize the single molecules, molecule overlapping is not shown graphically in Figure 1. However, some molecules do overlap (for instance, the molecules of type C-B-A share atom a_2).

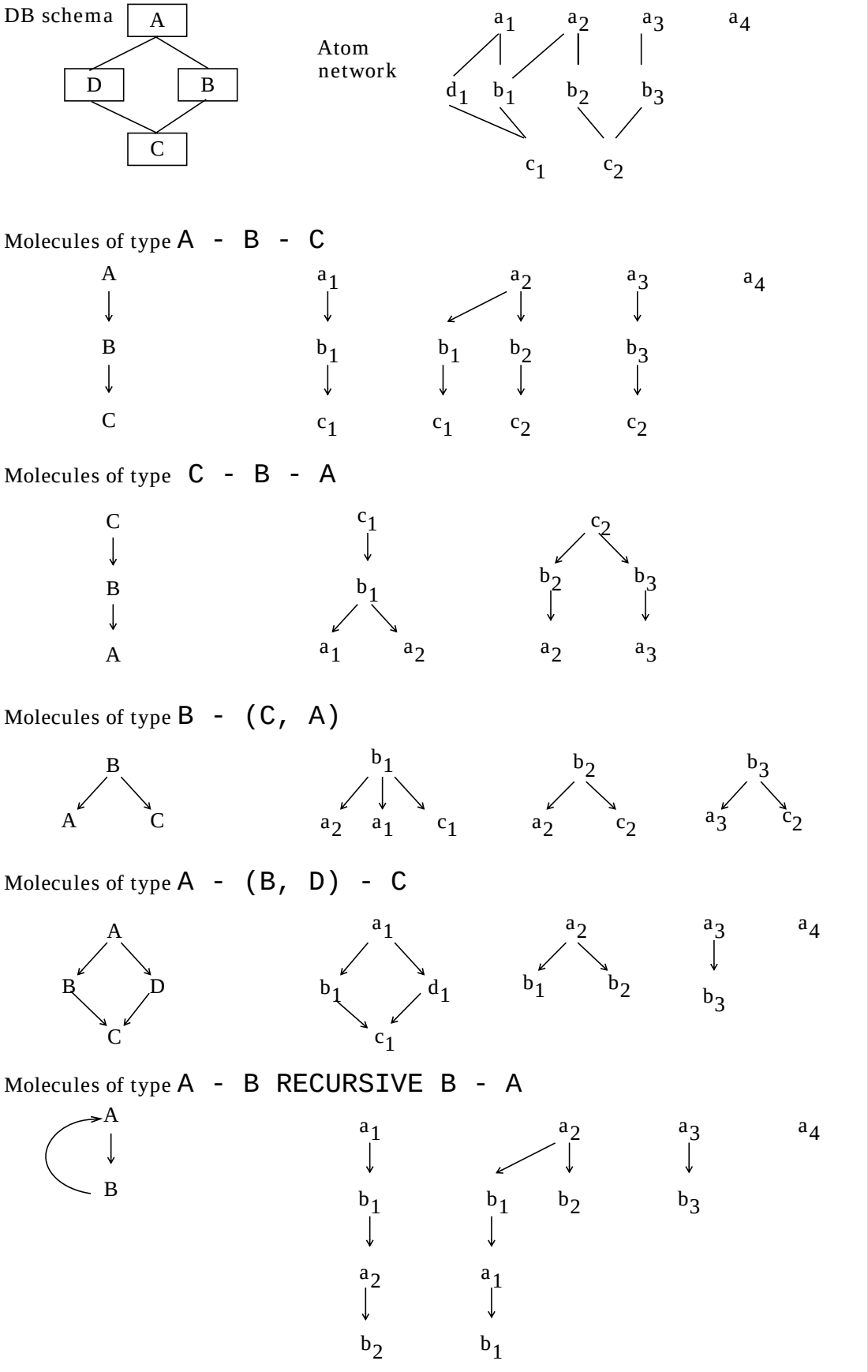
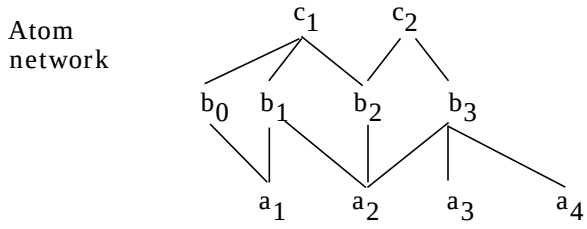


Figure 1: Example of molecules for various molecule type definitions

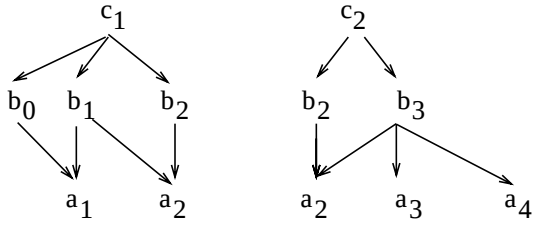
The following quantifiers may occur in a condition: EXISTS, EXIST_EXACTLY (n), EXIST_AT_LEAST(n), EXIST_AT_MOST (n), FOR_ALL.

The **projection clause** specifies which parts of the molecules should appear in the result. According to this clause, atoms or attributes are removed from the qualifying molecules. A value-dependent projection is possible ("qualified projection") and can be specified using a SELECT-statement with the special molecule type definition clause "RESULT". The last query in Figure 2 shows an example of a qualified projection. All molecules of type C-B-A belong to the result. Of these molecules, however, atoms of types *B* are included in the result only if there are exactly two links to atoms of type *A*. Atoms of type *C* are always included in the molecule, while atoms of type *A* never are.

The result of a SELECT query is a set of molecules of the type specified in the molecule type definition clause, which qualify under the conditions of the restriction clause and have undergone the projection specified in the projection clause. Figure 2 shows a sample atom network and some queries together with their resulting sets of molecules.



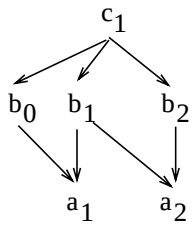

```
SELECT ALL
FROM   C-B-A;
```



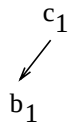

```
SELECT C
FROM   C-B-A;
```

c_1 c_2

```
SELECT C,B,A
FROM   C-B-A
WHERE  EXIST_EXACTLY (2) A;
```




```
SELECT C,(SELECT B
          FROM   RESULT
          WHERE  EXIST_EXACTLY (2) A)
FROM   C-B-A;
```



c_2

Figure 2: Some queries with their results based on a sample atom network

This short introduction to MQL neglects many aspects of our query language as well as of our data model [Mi88b, Mi89, Sch89]. Nevertheless, it will be sufficient to understand the discussions in the following sections.

3. The PRIMA Architecture

So far, we have sketched some features of the MAD model. In the following, we present a short overview of its implementation within the PRIMA system [HMMS87, Hä88]. Our implementation

model for PRIMA (Figure 3) distinguishes three different layers for mapping molecules, which are visible at the MAD interface, onto blocks stored on external devices:

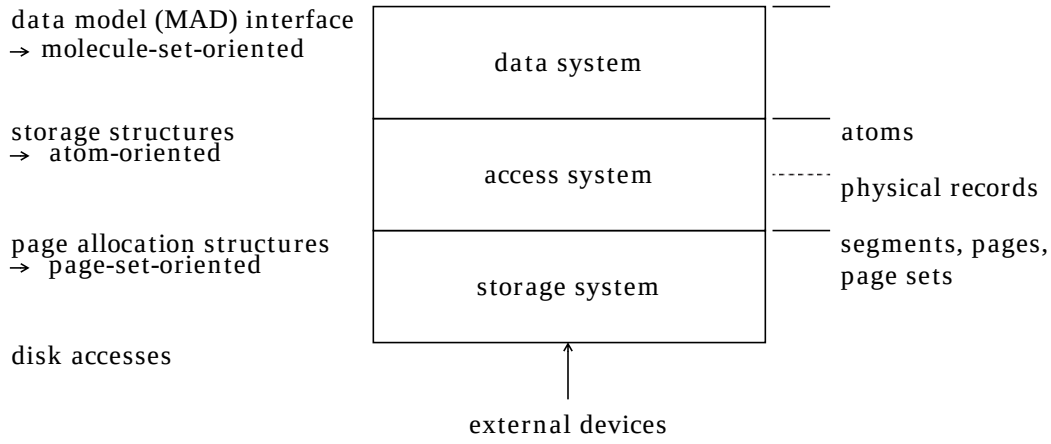


Figure 3: Implementation model of PRIMA

- The main task of the *data system* is to execute MQL statements. For this purpose, the user-submitted MQL statements are transformed into semantically equivalent query evaluation plans (QEPs). This step comprises compilation and optimization. Subsequently, these QEPs are evaluated, yielding the desired result (execution phase).

- The *access system* [Si89] provides an atom-oriented interface similar to the tuple-oriented interface of the Research Storage System (RSS) of System R [Ch81]. However, the access system is more powerful than RSS as outlined in the following. It allows for direct access and manipulation of a single atom as well as navigational atom-by-atom access to either homogeneous or heterogeneous atom sets. Manipulation operations (insert, modify, and delete) and direct access (retrieve) operate on single atoms identified by the value of their IDENTIFIER attribute.

  Different kinds of scan operations are introduced as a concept to manage a dynamically defined set of atoms, to hold a current position in such a set, and to successively deliver single atoms. The *atom-type scan* delivers all atoms in a system-defined order based on the basic storage structure that always exists for each atom type. Similarly, the *sort scan* processes all atoms according to a specified sort criterion thereby utilizing the basic storage structure. However, since sorting an entire atom type is expensive and time consuming, a sort scan may be supported by an additional storage structure called sort order. A sort order consists of a homogeneous atom set materializing a sort operator, i.e. a redundant copy of the atoms of one atom type, physically stored in the specified order. The *access-path scan* provides an appropriate means for fast value-dependent access based on different access path structures such as B-trees, grid files, and R-trees. The *atom-cluster-type scan* as well as the *atom-cluster scan* speed up the construction of frequently used molecules by allocating all atoms of a corresponding molecule in physical contiguity using a tailored storage structure called atom-cluster type [SS89]. An atom-cluster type corresponds

to a hierarchical molecule type. Thus, it clusters a heterogeneous set of atoms according to the directed type graph. Clustering is done in addition to the storage in the basic storage structure, i.e., the atom-cluster type is a redundant storage structure. Furthermore, clusters may overlap. Thus, an atom may be contained in several clusters of the same or another atom-cluster type.

- The *storage system* as the lowest layer of PRIMA pursues two major tasks: It manages the database buffer and organizes the external storage devices, thus being responsible for the data exchange between main storage and disk storage. For this purpose, the database is divided into various segments consisting of a set of logically ordered pages. All pages of a segment are of equal size, and can be chosen for each segment independently to be 1/2, 1, 2, 4, or 8 kbytes, being kept fixed during the lifetime of a segment. Thus, the page size may be adapted to the specific access pattern of the segment in order to diminish either the conflict rate or the number of I/O operations. The five page sizes, however, are not sufficient when considering the mapping process performed by the access system. Therefore, *page sequences* are introduced as predefined page sets supported by physical clustering. A page sequence is a set of logical consecutive pages of a segment that contain (from the viewpoint of the access system) a single object spanning these pages [DPS86].

This short overview of the PRIMA architecture should be sufficient as a basis for the following discussions.

4. Query Processing in the Data System

When an MQL statement is given to the data system, it is compiled into an equivalent QEP forming a directed graph. The vertices of this graph are labeled with operators, while its edges correspond to the data flow among the operators. The leaves of the QEP represent the operator *construction of simple molecules* (CSM), which builds up hierarchical, non-recursive molecules fulfilling some qualifications by using access system calls. The other vertices stand for operators such as *recursion*, *Cartesian product*, etc. Figure 4 shows a simple example of a QEP for a query containing a Cartesian Product (indicated by two molecule type definitions which are separated by a comma). In the resulting molecules, atoms of type D are omitted.

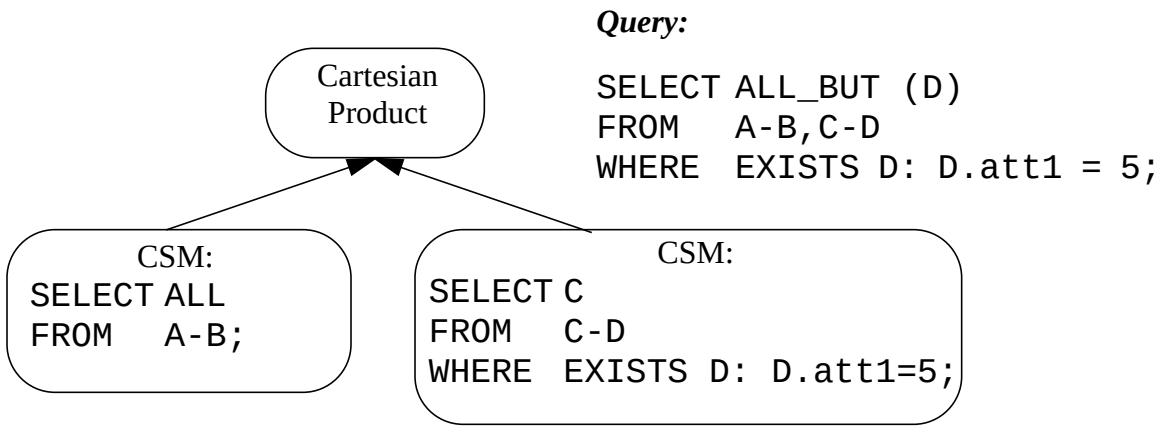


Figure 4: Sample query and corresponding QEP

The QEP may be subject to transformations by the optimizer, which is expected to generate a more efficient QEP by choosing appropriate access paths, selecting specific methods for each operator, and so on. Some of the possible choices for the CSM operator are discussed later.

When a QEP is executed, the operations indicated by its leaves are involved first, i.e., CSM operations are started. Whenever a molecule has been constructed, it is handed in a pipelined way to the next operator as indicated by the edges in the QEP. The root operator of the QEP produces the final result set of molecules [HSS88]. Each node of a QEP may be evaluated on another processor. Hence, the data driven approach chosen here saves inter-processor communication, since no control messages have to be passed. In Volcano [Gr90] there are OPEN, NEXT, CLOSE calls which manage a control-driven evaluation of a QEP as long as no processor boundaries are crossed. Otherwise, evaluation is done data-driven, but this fact is masked by a specialized operator. A consequence of this approach, however, is that parallelism has to be pre-planned, i.e. embedded in the QEP, which is not the case in PRIMA, where run-time assignment of operator evaluation and even migration of the corresponding tasks from one processor to another is possible.

All other operators get their input (directly or transitively) from the CSM operator in a pipelined way. Often, however, CSM is the only operator in a QEP, since in many cases the complex object facilities are sufficient to model the application's objects without explicit join or other higher operators. Therefore, it is necessary to direct one's attention to the optimization of the execution of the CSM operator. CSM constructs a set of molecules characterized by the following MQL query

```
SELECT  P(M)
FROM    M
WHERE   Q(M).
```

M is the definition of a non-recursive hierarchical molecule type, P(M) is a coherent subgraph of M (projection), containing the root atom type of M, and Q(M) is a restriction on M that can be deter-

mined by the consideration of a single molecule. As a consequence, Q(M) is not allowed to contain any sub-queries.

Conceptually, CSM scans the root atom type of M. For each atom of this type, CSM fetches the successor atoms according to M, then their successor atoms, and so on. When the whole molecule is fetched, it is checked against Q(M). If it qualifies, the molecule is postprocessed according to P(M) and added to the result set. This viewpoint, however, is only conceptual. In the following section we will show how CSM may be computed more efficiently.

5. Evaluating CSM

As mentioned above there are several algorithms to compute the result of a CSM operator. In the sequel we discuss some of the possible choices. For several sets of choices, we still lack criteria to determine the best one.

Two phase approach

Obviously, it would be inefficient to construct a molecule completely before evaluating the restriction, if only parts of the molecule are referenced in the corresponding expression. One should first consider a coherent minimal subgraph of M that is sufficient to decide Q(M). For this purpose, we introduce two phases of a molecule's construction: the *checking phase* checks the qualification of the molecule with respect to the restriction, the *completion phase* completes its components, i.e., reads those parts of the molecule that do not contribute to the result of the expression in the restriction clause. For example, for the query

```
SELECT ALL
FROM   A-(B,C)
WHERE  FOR_ALL C: C.Att1=5;
```

in the checking phase one would scan atom type *A*. For each atom *a* of type *A*, one then reads the atoms *c* of type *C* referenced by *a*, until one of them does not fulfill the condition `c.att1=5` or all are read and fulfill the condition. In the latter case, in the completion phase the atoms of type *B* referenced by *a* have to be read. In the completion phase, the order of accesses to atoms of the atom types that still have to be read does not matter, because all orders lead to the same amount of I/O. In the checking phase, however, the order of access is important.

Top-down or bottom-up

The condition Q(M) of the restriction clause may reference several atom types. Let us assume that Q has the following shape: Q (M) = Q_1 AND Q_2 AND Q_3 ... AND Q_n. In the checking phase, all these parts of the restriction clause have to be evaluated to TRUE if a molecule qualifies. For the evalu-

ation of a part Q_i of Q(M), only the atoms of a subset of the atom types have to be considered, i.e.atoms of the atom types mentioned in Q_i. We call this set of atom types R (Q_i).

One approach to evaluate the restriction clause is the *top-down approach*: All atoms of the root atom type are read. For each root atom, each part Q_i of Q(M) is considered. For each atom type in R (Q_i), the atoms referenced by the root atom type are read. If Q_i evaluates to TRUE, the next Q_i is considered. Note that the set of atoms which has to be taken into account is determined by the root atom's references. If the root atom type *R* does not directly reference an atom type *A* in R (Q_i), then all atoms of atom types on the path from *R* to *A* also have to be read. Obviously, a molecule may be skipped, if a Q_i evaluates to FALSE. Hence, an optimal order of the Q_i has to be found which minimizes the I/O effort needed to decide the disqualification of a molecule. This order cannot be determined by sorting the Q_i by their selectivity, because a very selective Q_i should not be put in the first place if one has to read the whole molecule to evaluate it.

If there is only one atom type in R (Q_i) a *bottom-up approach* may be more efficient. This approach starts with atoms of the only atom type in R (Q_i) that fulfill Q_i. These may be accessed with the help of an access path. From these atoms, references are followed towards the root atom type. For each atom of the root atom type which is identified in this way, the rest of Q(M) is evaluated as described before. The following query illustrates that the decision for either the top-down or the bottom-up approach is data-dependent:

```
SELECT ALL
FROM   A-B-C
WHERE  EXISTS C: C.att1=7;
```

Consider the atom network of Figure 5a. The atoms of type *C* with `C.att1=7` are marked by an arrow. Proceeding top-down leads to the construction of six molecules (reading atoms a1, b1, c1, a2, b2, c3, a3, b3, c4, a4, b6, c5, c6, a5, b7, a6, b8, b9, c7, c8, i.e. 20 atoms), but only three of them qualify. Starting from the *C* atoms and following the references bottom-up, however, leads to the three root atoms of the qualifying molecules (thereby reading atoms c3, b2, a2, c5, b6, a4, a5, b7, c6, i.e. 9 atoms if there is an appropriate access path on atom type *C*. Otherwise, *C* has to be scanned and another 5 atoms have to be read). In this case, all atoms which were read belong to the result. Consider the atom network of Figure 5b now, where the opposite is true: Top-down evaluation yields only qualifying molecules (reading 10 atoms), while bottom-up evaluation reads some atoms of type *B* and *C* that do not belong to the result (16 atoms are read if an access path exists, 21 otherwise). For the atom network of Figure 5a, a bottom-up evaluation is the better choice, while for the atom network of Figure 5b top-down is more profitable. In this example, there was only one Q_i, but in general there are several. If the root atoms qualifying with respect to one Q_i have been found, the remaining Q_i have to be checked for the corresponding molecules. Again, there is a choice of evaluating the conditions top-down (starting from the remaining root atoms) or bottom-up (computing the intersection of the sets of root atoms found). Note that a bottom-up evaluation

does not read all atoms of the considered atom types which may belong to a qualifying molecules. Hence, it has to be followed by a top-down completion of the molecule (In Figure 5a, atom b7 is read in the course of this completion).

A bottom-up evaluation is not possible if a molecule may qualify without having atoms of the atom type in R (Q_i) (structurally incomplete molecule). Hence, if a the qualification is quantified with one of the quantifiers `FOR_ALL, NOT EXISTS,` or `EXIST_AT_MOST` bottom-up evaluation is possible only if the database schema does not allow structural incompleteness for the atom type under consideration.

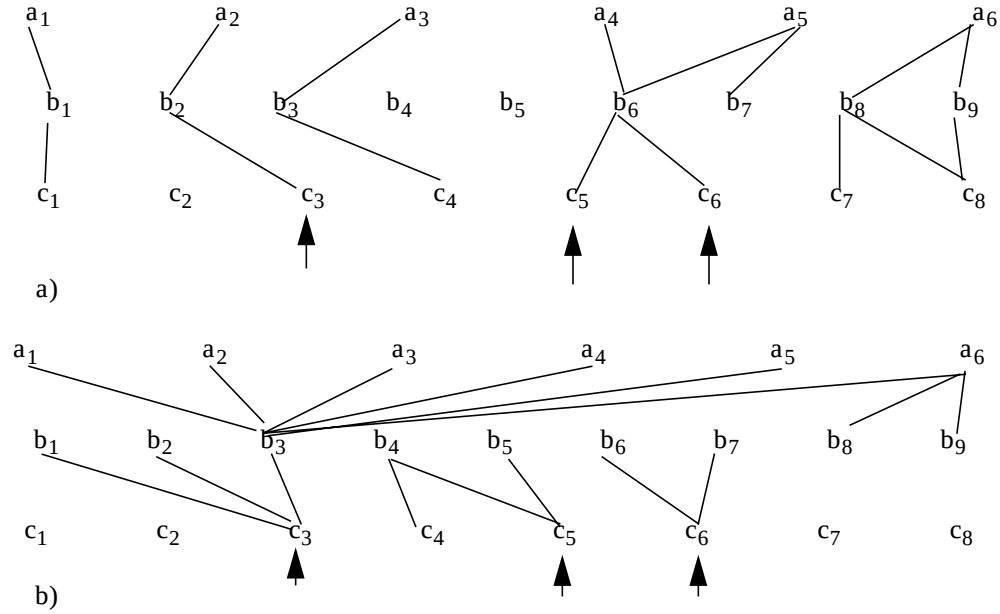


Figure 5: Sample atom networks

To summarize, one has to determine the optimal order of the evaluation of the sub-conditions as well as the strategy for the evaluation (bottom-up or top-down). Hence, we consider more strategies than just linear traversal [KKWD88]. For the cost computations one needs knowledge about the selectivity of the sub-conditions (i.e., how many molecules qualify with respect to the sub-condition) and the I/O effort needed to read the data needed to evaluate a sub-condition bottom-up or top-down. The selectivity depends on the network-structure of the data. Note that in the both network structures shown in Figure 5 the selectivity of the condition is different, although the same number of atoms of atom type *C* qualifies with respect to the condition `C.att1=7`. Obviously, the I/O effort also depends on the network structure. Therefore, it is crucial to have an accurate estimation of the number of atoms of a certain atom type which are directly or transitively referenced by atoms of another atom type.

How can this number be estimated? Note that in both atom networks shown in Figure 5 there is the same number of references from atoms of type *A* to atoms of type *B*, and from atoms of type *B*

to atoms of type *C*. Hence, knowing only the number of references linking two atom types is not sufficient for a satisfactory estimation. A more accurate estimation may be possible, if the transitive reference behavior is known (i.e., the number of atoms of atom type *A* transitively referenced by atoms of atom type *C* in our example). However, in a database with many atom types, this statistic obviously would explode in size. Furthermore, in our example there may be a correlation between the values of *att1* of atoms of type *C* and the number of atoms of type *B* referenced by these atoms.

Depth-first or breadth-first

Up to now, we have assumed that top-down evaluation considers one sub-condition after the other. However, the evaluation of several sub-conditions may be overlapped. We illustrate this by another example. The query

```
SELECT A
FROM   A-B-C
WHERE  EXISTS B: B.att1=7 AND FOR_ALL C: C.att2=8;
```

allows two different top-down evaluation methods. In both cases we start reading an atom of type *A* (one for each molecule). In the next step, we either read all atoms of type *B* and check them against the condition, and then check all the atoms of type *C* (breadth-first), or we read only one atom *b* of type *B*, check it against `B.att1=7`, then read the atoms of type *C* referenced by *b*, and check against `C.att2=8` before reading the next atom of type *B* (depth-first). The two alternatives resemble the "nested loop retrieval method" and the "sort domain retrieval method" [KKWD88]. Whether breadth-first or depth-first is the better choice depends not only on the selectivities of the predicates, but again on the correlation of attribute values and graph properties of the molecules.

Amount of parallelism to be used

The next problem arises from the fact that PRIMA is capable of parallelizing its actions. Consider the query

```
SELECT ALL
FROM   A-(B,C)
WHERE  EXISTS B: B.att1 = 8 AND EXISTS C: C.att1 = 9;
```

We could scan the atom types *B* and *C* in parallel, and check the condition for each atom. This parallelism leads to an improvement compared to the sequential version, if there are only few disqualifying molecules. For each disqualifying molecule, however, it is likely that more atoms of the molecule are read before the molecule construction is discarded than in the sequential version. We could specify every degree of parallelism between 1 (sequential) and the maximum. For example,

we could read two atoms of type *C* and three of type *B* in parallel, and then evaluate the condition. Again, we lack a criterion for the optimal degree of parallelism.

Use of atom clusters

As mentioned above, one can store heterogeneous sets of atoms in physical contiguity (atom cluster). Such an atom cluster corresponds to a molecule of a specific type, which must be hierarchical. In PRIMA, this is a redundant storage structure, i.e., the atoms belonging to an atom cluster are also stored in another storage structure (called the *basic storage structure*). Hence, one has to decide whether to use the atom cluster or not for the computation of a specific query.

Figure 6 shows a sample database consisting of atom types *A*, *B* and *C*, and the effect of an atom-cluster definition A-B-C. Some of the atom clusters overlap.

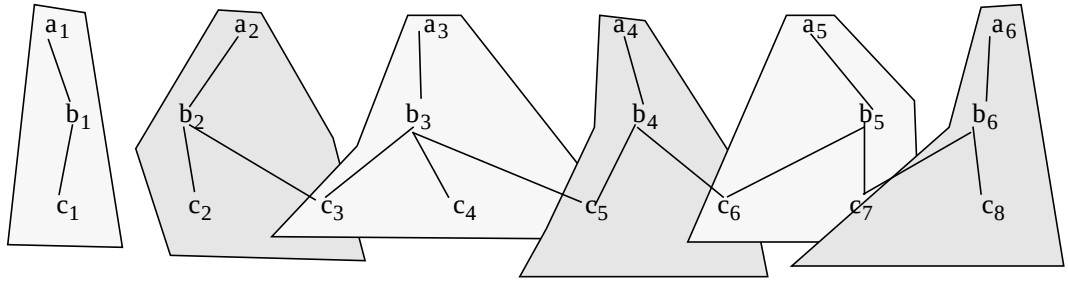


Figure 6: Atom clusters of type A-B-C

Consider now the following query:

```
SELECT ALL
FROM   A-B-C
WHERE  EXIST_EXACTLY (2) C: C.att1=3;
```

Obviously the atom-cluster type can be used to efficiently compute this query because we can immediately decide which atoms of type *C* belong to the molecule and hence decide the qualification condition. Compared to the top-down approach the use of atom-cluster types saves accesses to the external storage. Bottom-up evaluation is not appropriate. Because of the quantifier EXIST_EXACTLY it would identify a superset of the qualifying molecules which may be much larger than the result set. For the opposite direction of molecule definition the atom clusters cannot be used, as demonstrated in the following query:

```
SELECT ALL
FROM   C-B-A
WHERE  EXIST_EXACTLY (2) A: A.att1=3;
```

If in this query the quantifier is modified to EXISTS, the atom cluster type may be used to identify the root atoms of qualifying molecules. While in these cases the decision of whether to use the atom cluster or not are quite obvious, there are more difficult cases. One of them occurs, when the atom cluster is only partially covered by the query's molecule type, as in the following example:

```
SELECT ALL
FROM   A-B-D;
```

When the atom cluster is used, the atoms of *A* and *B* are read in one step, but also are the atoms of type *C*, which are not needed. It is not clear whether it is better to use the atom clusters or not. Similarly, the following query can be evaluated in several ways:

```
SELECT ALL
FROM   A-B-C
WHERE  EXISTS C: C.att1=6;
```

We could scan the atom clusters to find qualifying molecules, or we could apply the condition C.att1=6 to the atoms of type *C* (potentially supported by an appropriate access path), and then navigate bottom-up to the corresponding atoms of type *A*. Here, the decision again must be based on the graph properties in correlation with the attribute values. Additional one must be aware that in the case of an atom-cluster-type scan, some *C* atoms are read multiply because of the overlapping of atom clusters.

Note that the query containing the quantifier EXIST_EXACTLY could not be computed as easily by the second approach, because starting with a qualifying atom *c* of type *C* does not guarantee that the atoms of type *A* that are transitively referenced by *c* are root atoms of qualifying molecules.

In this section, we have considered several choices of algorithms that are available for the evaluation of the CSM operator. We saw that in many cases there are no obvious criteria that allow a decision among the algorithms at hand.

6. Conclusions

When considering the evaluation of queries in the molecule-atom data model, one has to focus mainly on an efficient computation of the CSM operator, because this is the basic operator that appears in every query. The CSM operator constructs hierarchical non-recursive molecules which qualify according to a restriction. Optimization problems mainly arise from the efficient evaluation of this restriction. The goal of optimization is to find a strategy which delivers the qualifying molecules with a minimal I/O overhead.

Standard optimization techniques (like those developed to optimize sequences of join operators) do not apply for CSM. While determining the root atoms of qualifying molecules may be done bottom-up, a top-down traversal has to follow in order to complete the molecule. Hence, in some cases on has to traverse a molecule type graph twice: bottom-up to find a set of root atoms (or a superset thereof) and top-down to complete the molecule.

The decisions required in the course of optimization are,

- determine a sequence of evaluation for the sub-conditions of the overall restriction,
- for each sub-condition, decide whether it should be evaluated top-down or bottom-up,
- decide which sub-conditions are to be evaluated in depth-first manner,
- find the optimal amount of concurrent (parallel) evaluation of sub-conditions,
- check for supporting redundant storage structures for the sub-conditions and decide about their usage.

We found that the decisions mentioned above heavily depend on the graph structure of the database. We also showed that in our case an abstraction of the graph properties in the form "*number of connections between two node types*", "*number of nodes of each type*", etc. is not sufficient. The kind of information really needed here is the dependencies between attribute values and graph properties.

This paper's purpose is to direct attention to the problems mentioned above, not to present the solutions found so far [Sch92]. In particular, there seems to be a need for a more accurate statistical model for graph structures. We believe that this is also a requirement for optimization in object-oriented database systems because the processing model of such systems as well as their data structures are quite similar to that of PRIMA.

We did not consider the other operators that may occur in a QEP of the PRIMA system. Some of these are "standard" operators (i.e., very similar to relational operators), and optimization mechanisms developed for other systems may be adapted. Among the more interesting operators is the *construction of recursive molecules*, which also can be computed by a huge variety of algorithms. Because of the differences in the definition of recursion between the MAD model [Sch89] and other systems, we have investigated this operator in more detail [Sch92].

7. References

Ch76 Chen, P.P.: The Entity-Relationship-Model - Toward a Unified View of Data, in: ACM TODS 1:1, 1976, pp. 9-36.

Ch81 Chamberlin, D.D., et al.: A History and Evaluation of System R, in: CACM 24:10, 1981, pp. 632-646.

DPS86 Deppisch, U., Paul, H.-B., Schek, H.-J.: A Storage System for Complex Objects, in: Proc. Int. Workshop on Object-Oriented Database Systems, Asilomar, 1986, pp. 183-195.

Gr90 Graefe, G.: Encapsulation of Parallelism in the Volcano Query Processing System, in: Proc. Int. Conf. on Management of Data SIGMOD '90, Atlantic City, NJ, 1990, pp. 102-111.

Hä88 Härder, T. (ed.): The PRIMA Project - Design and Implementation of a Non-Standard Database System, SFB 124 Research Report No. 26/88, University Kaiserslautern, 1988.

HMMS87 Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. 13th VLDB, Brighton, 1987, pp. 433-442.

HSS88 Härder, T., Schöning, H., Sikeler, A.: Parallelism in Processing Queries on Complex Objects, appears in: Proc. Int. Symp. on Databases in Parallel and Distributed Systems, Austin, Texas, 1988, pp. 131-143.

KKWD88 Kim, K.-C., Kim, W., Woelk, D., Dale, A.: Acyclic Query Processing in Object-Oriented Databases, in: Proc. 7th Int. Conf. on the Entity/Relationship Approach, Roma, Italy, 1988, pp. 193-210.

Mi88a Mitschang, B.: Towards a Unified View of Design Data and Knowledge Representation, in: Proc. 2nd Int. Conf. on Expert Database Systems, Tysons Corner, Virginia, 1988, pp. 33-49.

Mi88b Mitschang, B.: Ein Molekül-Atom-Datenmodell für Non-Standard-Anwendungen - Anwendungsanalyse, Datenmodellentwurf und Implementierungsaspekte, Ph.D. Thesis (in German), University Kaiserslautern, 1988.

Mi89 Mitschang, B.: Extending the Relational Algebra to Capture Complex Objects, in: Proc. 15th Int. Conf. on Very Large Data Bases, Amsterdam, 1989, pp. 297-305.

Sch89 Schöning, H.: Integrating Complex Objects and Recursion, in: Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases, Kyoto, 1989, pp. 535-554.

Sch92 Schöning, H.: Anfrageverarbeitung in Komplexobjekt-Datenbanksystemen, Ph.D. Thesis (in German), Universität Kaiserslautern, 1992.

Si89 Sikeler, A.: Supporting Object-Oriented Processing by Redundant Storage Structures, in: Proc. Int. Conf. on Computing and Information (ICCI'89), Toronto, 1989.

SS89 Schöning, H., Sikeler, A.: Cluster Mechanisms Supporting the Construction of Complex Objects, in: Proc. FODO '89, LNCS 367, Springer Verlag, 1989, pp. 31-46.