appeared in: Data and Knowledge Engineering 7, 1992, No. 3, pp. 181-200

Query Processing for Complex Objects

T. Härder, B. Mitschang, H. Schöning

University of Kaiserslautern

Erwin-Schrödinger-Straße
D-6750 Kaiserslautern
West Germany

Keywords: Query Processing, Complex Objects, Query Optimization, Data Model

Abstract

Over the last few years several new data models together with their languages have been developed to meet the increasing requirements of engineering or office applications. A major characteristic of these data models is their ability to process and manage complex objects which the relational model does not provide adequate support for. Whereas the problem of query translation for relational languages has provoked broad research activities during the last fifteen years, the analogous problem of translating non-procedural queries on complex objects into lower level programs for efficient execution has received only little attention.

This paper tries to reveal the new aspects of query translation and execution on complex objects as compared to similar activities when processing flat relations. For this purpose, we investigate the essential concepts necessary to perform compilation, optimization, and execution of queries on complex objects.

1. Introduction

Recently, the development of a new generation of database systems capable of supporting non-standard application areas such as engineering applications for CAD/CAM and VLSI or knowledge-based applications has emerged as an important direction of database systems research. These advanced applications differ from conventional (business) applications in a number of important aspects including data modeling and processing, concurrency control and recovery mechanisms, as well as access methods and storage structures. Most of the design and implementation approaches [5,6,8,25,26,28,36,41] refer to some kind of object-orientation and extensibility. In these cases, the overall uniting characteristic is adequate support for complex objects. This is accomplished in different ways starting from only a few selected extensions of the relational model and leading up to the integration and superposition of hierarchical structures on relations. Apparently, the provision of

- genuine and symmetric support for *network structures* (sharing of sub-objects in contrast to hierarchical structures, which are just special cases thereof), or even *recursive structures,*
- support for *dynamic object definition* in combination with
- *powerful*, yet *efficient manipulation* facilities

has drawn much less attraction, although it is urgently needed in many application areas for refined and accurate modeling as well as efficient processing of their objects (cf. for example [3] "... support for molecular objects should be an integral part of future DBMSs ...", where 'molecular' objects were classified according to their structure, leading to disjoint/non-disjoint and recursive/non-recursive complex objects). Especially for the support of such a complex-object (molecule) notion, we have designed the molecule-atom data model (MAD model) [30] and accomplished its prototype implementation PRIMA [17].

Whilst the current research interests tend to cover efficient and extensible processing as well as optimization of logical database languages and recursive queries, there is only minor research activity in the area of processing queries on complex objects. Thus, it is the scope of this paper to introduce some relevant concepts for such processing and to exemplify them by means of MAD queries in the PRIMA system. We want to identify problems arising from the facility to dynamically define complex object structures at query time, with respect to query optimization and query execution. The data model, used in this paper to show some intrinsic issues of complex object processing, could be classified as an extension to the non-first-normal-form models and other models (e.g. [27]) as well as to the relational model, which are all limited, at most, to hierarchical and statically defined complex objects. For reasons of convenience, the MAD model will be developed starting from the ideas of the relational model. This will provide a smooth and easy to understand introduction to the data model for the reader.

The prime focus of the paper is a comprehensive discussion of new aspects of query processing when complex as opposed to flat objects (relations) have to be dealt with. For this purpose, the MAD model and its implementation PRIMA are used as a reference example to identify theoretical as well as practical problems. Section 2 introduces the environment of our discussion, including system architecture and data model. In section 3, we present the three phases of query processing in our model, namely compilation, optimization, and execution as well as the novel issues related to complex object processing. In particular, major optimization challenges are shown. Section 4 presents some conclusions.

2. A Model for Query Processing

For our purposes, it is sufficient to refer to an abstract view of database processing in an analogous way to [10]. Most set-oriented database systems can be described by two major components: the *logical database processor* and the *physical database processor*. For example, in System R [2] the logical database processor is called the "Relational Data System" and the physical database processor the "Relational Storage System". The logical database processor translates the user queries into an internal representation called *query evaluation plan* (QEP), which is further optimized to guarantee efficient evaluation. At execution time, the physical database processor evaluates the previously generated QEP against the database in order to compute the requested result.

This quite abstract model of database processing will be taken as a basis for illustrating query processing for complex objects, as applied by the PRIMA system. This prototype database system reflects a multi-layered architecture with well-defined internal interfaces as a prerequisite for modularity, data independence, and extensibility in the various layers. For our purpose, it is sufficient to identify only the two layers shown in Figure 2.1. The external interface of PRIMA allows for handling of molecules and is defined by the MAD model. With regard to the PRIMA architecture it is a straightforward process to identify the two components performing logical and physical database processing: the PRIMA data system corresponds to the logical database processor and the PRIMA access system coincides with the (top-most layer of the) physical database processor. It is the task of the data system to perform the complex mapping of the molecule-set oriented external interface onto the atom-oriented interface of the underlying access system.

* * * Figure 2.1 * * *

In the following, we will characterize the data system introducing its upper and lower interfaces. Based on these descriptions, it is quite easy to explain and to illustrate the main concepts underlying the complex object processing in PRIMA.

2.1 The Molecule-Atom Data Model

The upper data-system interface coincides with the external interface of the PRIMA system and is established by the MAD model with its molecule query language MQL. MQL is embedded in a host programming language and can be directly used in an application programming environment; interactive operation is also supported. Both interfaces are equally powerful and are described in more detail in [15]. Here, we present an overview of the MAD capabilities for complex object management that is valid for both interface types.

From the Relational Model to the MAD Model

In the following, we presume that the reader is familiar with the relational model and its well-known concepts, e.g. tuples, relations, database and database schema, primary key, foreign key etc. To reach the level of the MAD model, we have to slightly modify our view of the relational model in the following way:

- Relations are named *atom types* and tuples are now termed *atoms*. In addition, atoms may have richer internal structures than tuples, e.g. multi-valued attributes.
- All relevant relationships between entity types, i.e. the foreign-key/primary-key connections between atom types, are explicitly specified in the schema and represented in the database.
- These relationships, simply called *link types*, are represented in a direct and symmetrical way. Thus, the database schema consists of undirected networks of atom types.
- Atoms may be connected to one another by *links* according to the link types specified in the database schema. Hence, a database can be seen as an undirected network of atoms.

Thus, the MAD model uses atoms as a kind of basic elements to represent real world entities. Similar to a tuple, an atom consists of attributes of various data types, is uniquely identifiable, and belongs to its corresponding atom type. The attributes' data type can be chosen from a richer selection than in conventional data models. Here, the type concept has been extended by RECORD, ARRAY, and the repeating-group types SET and LIST to yield a powerful structuring capability at the attribute level. For the realization of links between atoms, we have introduced two special types. The IDENTIFIER type serves as a surrogate, which allows for atom identification. Based on this type, it is easy to define the REFERENCE type providing a list of identifier values belonging to atoms of exactly one atom type.

* * * Example 2.2 * * *

A link type (e.g. between atom types A and B) is represented by a pair of REFERENCE attributes, one in each atom type involved (e.g. attribute b of A and a of B in Example 2.2). As syntactical suger, we denote this link type A-B, if there is only one link type between atom types A and B. A REFERENCE attribute cannot exist on its own, but always has a corresponding 'counter' REFERENCE attribute. A link between two atoms (e.g. between atoms a1 and b1 of Example 2.2) is represented by according values of the REFERENCE attributes forming the link type (in Example 2.2, the value of attribute b of a1 contains the value b1, and the value of attribute a of b1 contains a1). Obviously, all kinds of relationships (1:1, 1:n, n:m) can be directly mapped by this concept. This direct representation and the consideration of bidirectional links establish the basis of the model's flexibility. Moreover, atom-type crossing operations along these links are more efficient than joins in the relational model due to direct (n:m)-relationship representation and system-controlled surrogates. In Example 2.2, three atom type definitions are shown (for atom types A, B, and C, respectively). The cardinality of the REFERENCE attributes is restricted by a minimum and a maximum number of identifiers per attribute value. For example, each A atom has to have at least 2 references to B atoms (there is no upper limit), whereas each C atom must have at most 5 references to B atoms, and at least 1. The two link types correspond to n:m relationships. The resulting atom type network is depicted in Example 2.2. Furthermore, a sample database corresponding to this schema is shown.

Based on the atom networks, the model's complex objects (*molecules*) are dynamically definable as higher level objects which are viewed as structured sets of interconnected and possibly heterogeneous atoms. Their structure is described by a directed connected sub-graph of the database schema, whose nodes are the atom types involved (e.g. A, B, and C in Example 2.3) and whose edges are the link types to be used (A-B and B-C in Example 2.3). This graph must have one designated node (the *root*) from which all other nodes can be reached. The corresponding atom type is called *root atom type*. The structure graph is allowed to be cyclic only in case of recursive molecules (e.g. bill-of-material). For each molecule structure, there exists a corresponding *molecule set*, which groups all molecules showing the specified structure. At least from the conceptual point of view, the dynamic derivation of the molecules proceeds in a straight-forward way using the molecule structure as a kind of template, which is laid over the atom networks. Thus, for each atom of the root atom type one molecule is derived following all links determined by the link types of the molecule structure until the leaves are reached (In Example 2.3, the molecule construction starts from atom type A, then following link type A-B and B-C. For example, starting with atom a1, the atoms b1, b2, and b3 will be added to the molecule. In the next step, c1, c3, c2, and c4 will also be included into the molecule). This process is termed *hierarchical join* [32]. Both, the molecule structure together with its derived molecule set are denoted *molecule type*. The basic means used to tailor a molecule type appropriately are the well-known selection and projection operations. They are applicable to each molecule type.

The flexibility of the MAD model stems from the fact that the same database (i.e. atom networks) can be used to derive totally different molecule types, just by specifying different molecule structures. Example 2.4 shows a bunch of molecule structures that are valid for the same database consisting of the atom types A, B, C, and D as well as the link types in between. The reason that this works well, lies in the direct and bidirectional link concept allowing for a symmetrical use of the database.

* * * Example 2.3 * * *

Query and Manipulation Facilities in MQL

The operational power of the MAD model is founded on its adequate means for molecule processing provided by MQL. Similar to SQL [46], MQL is subdivided into three parts reflecting data definition (DDL), load definition (LDL), and data manipulation (DML). Here, we focus on the latter, that is, on query (i.e. retrieval) and manipulation (i.e. insertion, deletion, and modification) capabilities.

Analogously to SQL, there are three basic language constructs:

- The *FROM* clause specifies the molecule type to be worked with.
- The *WHERE* clause allows for the restriction of the corresponding molecule set.
- The projection clause (i.e. the *SELECT* clause in the case of retrieval statements) defines the set of the molecule's atoms to retrieved and is responsible for proper molecule projection.

Compared to SQL, these constructs exhibit extended semantics and syntax according to the more complex objects which have to be dealt with. They form the basis of all DML-statements offered. The result of each query is also a molecule type. Thus, it can be shown [31] that the closure of the MAD model under its molecule operations is guaranteed. This is a very important fact, which allows for the nesting of molecule queries; each molecule-type specification (e.g. A-B-C in Example 2.5) can be replaced by a molecule query (cf. Example 3.3).

In the following, we wish to illustrate the descriptive and operational power of the MQL-DML in more detail, thereby refining the basic clauses, which are depicted in Example 2.5 and introduced above. The *FROM* clause of each given DML-statement determines the molecule structure to be operated upon.

* * * Example 2.4 * * *

There are two generic kinds of molecule structure (cf. Example 2.4):

- The molecule structure of *network-like molecule types* (cf. Example 2.4a) resembles a meshed graph. In this case, there may be component types with more than one link type. In graphic terms, this fact is expressed by nodes with more than one incoming edge - of course, a hierarchical graph is just a special case thereof. All molecules of the corresponding molecule set have to obey the associated network semantics: During molecule construction only those atoms are selected as part of that molecule for which there is a link from already selected atoms for all incoming edges. Thus, it is guaranteed that all constructed molecules are built up from a set of atoms that are interconnected according to the specified molecule structure (cf. Example 2.4). Since it is possible to define molecules having non-disjoint atom sets, which exhibit a general graph structure, the model allows for the sharing of sub-objects between molecules in a natural way.
- *Recursive molecule types* (cf. Example 2.4b) use a network-like component molecule type combined with a recursion-defining link type expressed in a special *RECURSIVE* clause (e.g. C-A in Example 2.4b). The resulting molecule structure is the recursively continued molecule structure of its component molecule type. The derivation of the corresponding recursive molecules has to be performed step by step in an iterative manner, going from one level (i.e. component molecule) to the next subordinate level using the recursion-defining references (cf. Example 2.4b). Here, the transitive closure has to be computed, which could be additionally cut off by an optional restriction clause (*UNTIL* clause). A more detailed description of the recursion facilities of the MAD model can be found in [39]. By means of recursive molecule types, we are able to construct molecules exhibiting a dynamic number of nesting levels which contributes a major enhancement compared to the static number of nestings found in non-first-normal-form tuples. Thus, the MAD model is capable of handling recursive molecule types which are defined as true data model objects in contrast to a number of other data models, e.g. [1,27,42].

Although molecule types are generally defined as part of a query, it is possible to *predefine* frequently used molecule types and to assign a name to them. This is similar to a view definition in the relational model.

The optional *WHERE* clause restricts the molecule set (determined by the molecule type of the *FROM* clause) to those molecules satisfying the given qualification condition. Since molecules normally comprise of an interconnected heterogeneous set of atoms, it is necessary to extend the qualification facilities of the language. Thus, it should be possible to query the molecule structure yielding *quantified qualification* terms. Hence, testing for the existence (*EXISTS*-quantifier) of atoms of a given component type or using the *FOR_ALL*-quantifier as an alternative quantification construct is allowed. There are also the specialized quantifiers *EXISTS_AT_LEAST* n, *EXISTS_AT_MOST* n, and *EXISTS_EXACTLY* n. The standard presetting used in MQL is the existential quantifier, so that the use of quantifiers is optional. The query of Example 2.5 depicts an explicitly quantified qualification condition.

The well-known projection expressed by simply listing the components (atom types with their attribute types) to be retrieved is also valid in MQL. To retrieve the whole result set in an unchanged state the keyword *ALL* may be used. The complementary *ALL_BUT* construct allows to list components that are not to be retrieved. Furthermore, for more selective specification of the resulting molecules, MQL introduces the so-called *qualified projection* (complementary to the above mentioned *unqualified projection*). Qualified projection is expressed as a *'SELECT...FROM...WHERE'* expression within the projection clause. This nesting allows for a supplementary projection of the components of the result-set molecules by evaluating the qualification condition of the *WHERE* clause within the qualified projection. The scope of this qualification comprises the whole molecule; therefore, we use the presetting *RESULT* for the corresponding *FROM* clause. Referring to our Example 2.5, only those C atoms are finally retrieved, which satisfy the qualification term stated. Exploiting these two projection capabilities, we are able to retrieve only those components (sub-molecules) of the result-set molecules we are interested in. Hence, the projection clause determines the final structure of all molecules in the result set. In the case of retrieval, the *SELECT* clause may be extended by an *order specification*. Furthermore, *aggregation functions* like *SUM* and *AVG* can be applied.

* * * Example 2.5 * * *

Comparison to Other Models

After having sketched the MAD model and its language MQL, it is worthwhile to draw a comparison to other models and their languages. A rough but expressive comparison can be done just by looking at the different complex object concepts supported: It is obvious that the MAD model with its support for network structures comprises all models that are based on flat or only hierarchically structured objects. Thus, the relational model, the extended relational model [27], and even the non-first-normal-form models [1,25,42] are just special cases thereof. A more detailed comparison is possible using the models' formalizations: comparing the molecule algebra [31] defined for the MAD model with the NF^2 relational algebra [42] or other NF^2 approaches [35,4,38] leads to the same conclusions.

Furthermore, the MAD model is able to deal with recursive complex objects as data model objects showing a dynamic number of levels in the object's structure [39]. This gives the MAD model more flexibility compared to the above mentioned models that only support a fixed nesting structure and are not able to represent the result structure of recursive queries within real data model objects. Meanwhile, there are attempts to enhance these models to include a recursion facility [23,4].

Another major issue of the MAD model is its provision for dynamic object definition. That is, the molecule types to work with are defined in the query language and are not statically fixed in the database schema as it is the case in most NF^2 approaches. For this reason, the MAD model offers a great flexibility in complex object definition and management (e.g. all molecule types sketched in Example 2.4 might have been defined over the same database).

Obviously, every attempt to compare the models' query languages is strongly influenced by their underlying complex-object concepts. MQL is an SQL-like language offering set-orientation and expressiveness with respect to dynamic object definition, powerful molecule restriction, and an extended projection facility. Thus, the most important concepts expressible in SQL-like languages [27,25] for other data models (offering less complex object concepts) are also expressible in MQL.

A language approach along the lines of MQL has been described in [41]. There, the language CERMoQL offers a set-oriented, declarative access to the database objects defined by an extended Entity-Relationship data model [10]. Support for structured (including recursion) and versioned objects is given.

Considering object-oriented data models [8,28], we firstly have to notice that they are mostly characterized by their facilities comprising modeling and managing of meshed and sometimes even recursive structures, which are frequently viewed from different points, depending on the current processing state. The link concept and the concept of dynamic molecules combined with the expressiveness of MQL seem approximately equal to these characteristics defining object orientation. In contrast to these models, MAD does not support behavioral object orientation. The early available object-oriented database models [7] show (compared to e.g. relational models) some deficiencies in their query capabilities, i.e. set-orientation was mostly out of their scope. The database was queried in a navigational manner along the references defined between the database objects (e.g. path expressions in OPAL [33]). By now a lot of research focuses on full-fledged query languages for object-oriented database models [22, 44]. Seen from a more general point of view the upcoming object-oriented query languages exploit facilities to query along predefined relationships (in [6] called functional join) as well as to use traditional value-based relational join capabilities. The functional join facility resembles the molecule building concept of MAD, and the general/relational join capability is also available in MAD/MQL. Therefore, there seems to be on one hand a considerable overlap of language concepts between MAD/MQL and object-oriented query languages that lead on the other hand to similar processing concepts. This argumentation done for MAD/MQL, as a representative of a complex object data model, seems to carry over to other complex-object data models [6, 45]. Of course, more detailed work needs to be done on the finer-grained similarities and differences between both camps.

After having described the MAD model, which forms the upper interface of the data system (Figure 2.1), we now introduce the interface of the access system, which provides atoms as the basic building blocks for molecules.

2.2 The Access System Interface

The access system offers an atom-oriented interface which allows for navigational retrieval and modification of atoms. To satisfy the retrieval requirements of the data system, it supports *direct access* to single atoms as well as atom by atom access to homogeneous and heterogeneous atom sets.

Manipulation and direct access operations refer to atoms identified by their logical address. The logical address (or surrogate) is used to implement the IDENTIFIER attributes as well as the REFERENCE attributes.

Scans are a concept to control a dynamically defined set of atoms, to hold a current position in such a set, and to successively deliver single atoms or only selected attributes thereof for further processing.

appeared in: Data and Knowledge Engineering 7, 1992, No. 3, pp. 181-200

The result set of the scan can be restricted by a simple search argument (or some additional start/stop conditions in the case of access paths) solvable on each atom. Some scan operations depend on the existence of a certain storage structure, which is generated by corresponding LDL-statements. The PRIMA access system supports the following scan operations at its interface:

- the atom-type scan based on a general basic storage structure,
- different access path based scans (e.g. a scan based on B-trees)
- scans guaranteeing a certain sort order, which may either be materialized or dynamically derived.
- the atom-cluster scan which operates on clusters of heterogeneous atoms.

Whereas the first three scan types support "horizontal' access to a homogeneous atom set belonging to one atom type, the last one allows for the 'vertical' access to a heterogeneous atom set across several atom types. The concept of atom clusters [43] has been introduced to speed up construction of frequently used molecule types. All atoms of the corresponding molecules are stored in physical contiguity, i.e. the molecules are pre-derived and materialized in this storage structure.

3. Concepts for MQL Processing

In the following, we concentrate on concepts for efficiently mapping the molecule-set oriented MAD/MQL interface onto the atom-oriented access system interface. This task is accomplished by the PRIMA data system.

DML-statements are expected to be used more than once, since they are normally embedded in application programs which are executed quite frequently. This means that the overhead for repeated executions should be minimized. A first way in which to do so is to separate compilation from execution and to store the compilation result within a so-called *access module*. Thus, repeated execution does not require repeated compilation. Secondly, query optimization is mandatory to make execution more efficient. This leads to three phases of DML processing:

1. *Compilation* of the DML-statement generates an executable, but not necessarily optimal query evaluation plan (QEP) and stores it within an access module.
2. *Optimization* transforms the QEP and replaces the generated access module. This includes rearrangements within the QEP, strategy choices, and selection of access paths.
3. *Execution* of the access module requests atoms retrieved by the access system and combines them in order to build up the result set. Execution can be repeated independently of phase 1 or 2.

Although these three processing phases are common to other well-known query processing approaches, we have to consider several novel aspects due to the inherent properties of molecule processing (i.e. dynamics in molecule definition, atom heterogeneity in molecule building). These features will be highlighted in the subsequent sections.

appeared in: Data and Knowledge Engineering 7, 1992, No. 3, pp. 181-200

3.1 The Compilation Phase

The compilation phase accepts only correct MQL-statements and generates a semantically equivalent QEP. For this purpose, the query is checked to see whether it fulfills all constraints imposed by the MAD model and the database schema, e.g. whether all names are defined, whether the result is described by a coherent graph, and so on. Furthermore, the compilation phase performs a first step towards query optimization as introduced by [20], i.e., “standardization” (which means to change the query to a standardized form).

Standardization

To achieve this standardization, we have to look through all clauses of an MQL statement:

- ***Projection clause:*** Since MQL allows for the use of *ALL* and *ALL_BUT* in *SELECT* clauses in the place of an exhaustive attribute enumeration, these keywords have to be replaced by their actual meaning, i.e., by a set of attribute names.
- ***FROM clause:*** Predefined molecule names may be used in the FROM clause to specify the scope of the query. In this case, the underlying molecule definitions have to be substituted. This process has to be repeated recursively, because molecule definitions can be based on other molecule definitions.
- ***WHERE clause:*** Besides the well-known standardizations of boolean expressions (cf. [20]), there is an MQL-specific one: MQL allows for incompletely quantified expressions in *WHERE* clauses (with each non-quantified expression E containing attribute “att” of atom type AT having the semantic “*EXISTS* a $\in$ AT: E”). To standardize the representation, *expression completion* forms the existential closure [29].

After the compilation phase is finished, a correct QEP has been generated and stored within an access module, which already may be executed by the query execution components. We have designed QEPs in such a way that they do not need to undergo any optimization in order to be executable, although optimization is strongly recommended for complex retrieval statements. As a consequence, associated with each operator in a QEP, there is a standard execution technique to be applied (e.g. nested loop strategy for two-way join), which may be changed by the optimization phase; access paths are not considered within the compilation phase and the building of the initial QEP. In the following, we show the structure of a QEP for retrieval statements.

Representation of QEPs for retrieval statements

Retrieval QEPs consist of an *operator graph* (cf. Example 3.1) describing the execution plan. Evaluation of a node operates on its children's results. The left child's result may be computed immediately, whereas those of the others are prepared either concurrently or later, depending on the node's type and the evaluation strategy applied.

* * * Example 3.1 * * *

Generally, it is possible to divide all nodes of an operator graph into two classes (cf. Example 3.1). The *leaf nodes* are used to construct the molecules, whereas the *inner nodes* (projection, recursion, aggregation, etc.) operate on the derived molecules (typically in main memory). All leaves are of type CSM ("construction of simple molecules"). This operator type represents the class of syntactically and semantically correct queries of the following form:

SELECT	<unqualified projections>
FROM	<one non-recursive, hierarchical molecule type>
WHERE	<molecule qualification Q>

As already mentioned, the inner nodes are operators that act on the molecules delivered by their children (leaf to root evaluation). Some examples of these operators are:

- Aggregation of selected values (provided by the only child) by a function like SUM, COUNT, AVG. In Example 3.1, an aggregation is used to compute one set of identifiers from a set of molecules.
- Qualified projection of sub-molecules (right child) by conditions referring to the molecules delivered by CSM (left child). The results of qualified projection are always passed on to aggregation in order to compute the new reference list to each atom of the root atom type defined by the qualified projection (where qualified projection may have cut off some sub-molecules).
- Construction of recursive molecules using non-recursive component molecules. Each recursion level is constructed separately (using CSM). The component molecules of the next recursion level are determined by the recursion-defining reference attributes, together with the termination condition (UNTIL clause) and the general recursion termination rule for processing the transitive closure (fixed point semantics [29] w.r.t. the set of non-recursive component molecules). This level by level evaluation is shown in Example 2.4b for a simple recursive molecule.

Example 3.1 combines several operators to create an operator graph. First of all, the left-most operator (construction of simple molecules) has to be evaluated, delivering molecules of the hierarchical molecule type A-B-C which satisfy the qualification criteria given. Based on this result, the qualified projection of the sub-molecules has to be performed. For this purpose, further operators are required: The (right) "construction of simple molecules" selects from the result set of the first (left) CSM operator all sub-molecules to be projected and the aggregation operator computes the valid reference list to structurally reflect the proper qualified projection. In our example the qualification of the sub-molecules has to be supplemented by a parameter value (value 1) derived from the "surrounding" molecule (cf. operator "qualified projection" in Example 3.1). From the operator graph, an access module is generated which employs standard access methods. This access module is modified in the optimization phase to incorporate more appropriate access methods. Hence, for very simple statements (e.g., insertation of single atoms based on constant values) the optimization phase may be ommitted.

Like the MQL-statements for manipulation and retrieval, manipulation and retrieval QEPs resemble each other. Since more complex language constructs are applicable for retrieval statements, QEPs for retrieval (*SELECT* statements or sub-queries) are more complex than those for manipulation statements, and therefore are more likely to be far from the optimal QEP. For these reasons, we discuss optimization only for SELECT statements in the following. Nevertheless, the optimization phase can also be applied to manipulation statements using similar techniques.

3.2 The Optimization Phase

According to [20], the next steps of query optimization are simplification and amelioration (corresponding to what [11] calls query modification). Finally, query refinement (called "query optimization" by Freytag [11]) generates a set of access plans, the cheapest of which is chosen to be executed (or in our case to be stored in an access module).

Simplification and amelioration

We first describe the simplifying transformations which are typical for the MAD model. The various simplification techniques developed for the relational model are usually also applicable for MQL queries.

- Molecule structure definitions may contain components that are neither projected nor used for qualification. This case will occur quite often when predefined molecule types are used. If those components are leaves of a molecule graph, they can be cut off recursively without influencing the result (Example 3.2).

SELECT	A(att1), B(att2)		SELECT	A(att1), B(att2)
FROM	A-B-C-D-*E*	≡	FROM	A-B-C-D
WHERE	EXISTS D:(D.att3=7);		WHERE	EXISTS D:(D.att3=7);

C and E are components that are neither projected nor used for qualification. E can be cut off, since it is a leaf of the molecule graph. C must not be removed, because it is needed to access the D atoms needed for qualification.

<u>*Example 3.2:*</u> Molecule structure simplification by cutting off unused components.

- As each MQL query delivers a set of molecules, MQL consequently allows for the use of a query at any place where a molecule type definition is allowed, thus leading to query nesting in *SELECT, FROM* (cf. Example 3.3), and *WHERE* clauses. Some nested MAD queries can be transformed into equivalent MAD statements with a lower degree of nesting (*statement simplification*, Example 3.3) based on well-known strategies, as described in [21,13].

SELECT	A(att1, att2)		SELECT	A(att1, att2)
FROM	*(SELECT A (att1, att2, att3)*	≡	FROM	*A-B*
	FROM A-B		WHERE	*(B.att2 = 7)* AND (A.att3 = 9);
	WHERE B.att2 = 7)			
WHERE	A.att3 = 9;			

<u>*Example 3.3:*</u> Statement simplification by sub-query elimination.

- Qualified projection is supplied to select components by their values rather than by their types. Qualifications on whole molecules are normally expressed using the *WHERE* clause. Thus, qualified projection concerning the root atom type can be equivalently transformed into a qualification attached to the *WHERE* clause.

appeared in: Data and Knowledge Engineering 7, 1992, No. 3, pp. 181-200

```
SELECT (SELECT A (att1, att3),B(att1)       SELECT  A(att1, att3), B(att1)
        FROM   RESULT                  ≡    FROM    A-B
        WHERE  A.att2 = 7)                  WHERE   (A.att2 = 7) AND (A.att3 = 9);
FROM A-B
WHERE  A.att3 = 9;
```

Example 3.4: Elimination of qualified projection on the root atom type.

- Predicates hidden implicitly in the FROM clause are made explicit (*restriction enhancement*). The query shown in Example 3.5 does not contain an explicit restriction on atoms of type A. However, there is a restriction on atoms of type B, which only can be fulfilled, if there are any B atoms, i.e. if the REFERENCE attribute b OF A (pointing to B atoms) is not empty. Hence, the restriction ***A.b <> EMPTY*** is implicitly hidden in the query, and is made explicit as shown in the right part of Example 3.5.

```
SELECT   A(att1), B(att2)            SELECT A(att1), B(att2)
FROM     A-B                    ≡    FROM   A-B
WHERE    EXISTS B:(B.att3=7);        WHERE  EXISTS B:(B.att3=7) AND (A.b <> EMPTY);
```

Example 3.5: Restriction enhancement

Query Refinement

Finally, alternative strategies for the execution of the operators in the QEP have to be considered in order to find the cheapest execution plan. In the following, we argue why we concentrate on only one operator (CSM) in the subsequent discussion. Of course, our optimizer has to cope with all types of operators. Some of them may be handled very similar to relational operators. For example, selection and (unqualified) projection do not pose many new problems. Therefore, we concentrate on join and related problems.

For example, besides the hierarchical join used in the molecule structure definition, MAD allows for a traditional value-based relational join. Because of the similarities, optimization techniques for this operator can be derived from those developed for the relational join. Nevertheless, this kind of join is not a typical operation on molecules. In most cases, those atom types which are expected to be used together in one query will be connected by link types and hence do require the hierarchical join as mentioned before.

CSM Optimization

Besides this observation, we focus our discussion on the CSM operator for the following reasons:

- Since each leaf of a QEP must be of type CSM, this operator appears in every QEP. Thus, its optimization improves the performance of all queries.
- CSM is the only operator which (using the access system) directly reads atoms from the database. Therefore, the problem of access path selection appears only here.
- Many operators just work on the result of CSM (typically in main memory) without performing very complex computations. Therefore, their optimization is not a primary issue.

- Construction of recursive molecules (CRM) also is a critical operator. Basically, it consists of a combination of non-recursive molecules, constructed by CSM. Thus, it also takes advantages from CSM optimization. To discuss the applicability of more enhanced optimization techniques developed for recursive queries to CRM goes beyond the scope of this paper.

Specific Optimization Problems of CSM

Although CSM performs a specialized join operation ("hierarchical join" [32]) and evaluates conditions on the result, its optimization differs from optimization in the relational model in various aspects:

While the selection of join orders and join methods is a main task in the relational model [34], these problems partially disappear for CSM, because the molecule structure defined in the FROM clause strongly reduces the number of meaningful orders for the refinement. The basic operation which is used to build up molecules is a join operation ("hierarchical join") with specific properties:

- Only atom types which are connected by a link type may be combined by a hierarchical join. Hence, the number of possible join sequences decreases compared to the relational join.
- Our hierarchical join is an n:m join, i.e. each atom a of type A within a molecule may have several descendants d_i of type D (besides the descendants of other types) which may be shared with other atoms of type A. The join condition is "ID of d_i is contained in a.rd", where rd is a REFERENCE attribute of A pointing to atoms of type D. Thus, a sort-merge strategy cannot be applied. Instead, we use a nested loop algorithm of the following form:

  Foreach atom a of type A
  Foreach entry e in a.rd
  Call all atoms d of type D from the access system via condition d.ID = e

  Note that "Find d with d.ID = e" is a very efficient operation, by far faster than any value-based restriction on atoms, because the access system is optimized for an identifier-based access. This is the reason why we do not use the semantically equivalent form:

  Foreach atom a of type A
  Call all atoms d of Type D from the access system via condition a.ID IN d.ra,

  where ra is the REFERENCE attribute pointing from D atoms to A atoms.
- In contrast to the relational join, the result of a hierarchical join does not consist of a set of tuples, but of a set of molecules, i.e. a set of structured sets of atoms. Hence, we have to consider the case, where an atom is shared among several molecules of a result set or is descendant of more than one atom within a molecule. In either of these cases, it will not be redundantly contained in the data output (cf. Example 2.4). As a consequence, the algorithm for the hierarchical join has to be refined as follows:

  Foreach atom a of type A
  Foreach entry e in a.rd
  If atom d with d.ID=e is not yet contained in the result's atom set
  call d from the access system via condition d.ID = e

Note that the result's atom set may contain atoms of different atom types.

- The asymmetry of the hierarchical join should be stressed. If building a molecule of type A-D, each A atom will be included in a molecule, even if it has no D descendants. On the other hand, a D atom which does not have an A ancestor will not belong to any molecule of the type A-D. A relational join would exclude both the A and the D atom. This is a reason, why the reduction of the size of intermediate results is not a primary optimization objective for CSM. There are no tuples which fall aside because they do not fulfil the join condition when building a molecule top-down. This is independent of the order in which atoms are joined to the molecule, as long as it is compatible with the semi-order imposed by the direction of links in the molecule. Some optimization algorithms for CSM therefore resemble to those developed for the network model [9].

Restriction evaluation

CSM does not only perform the hierarchical join, but also evaluates restricting conditions on the resulting molecules. Hence, we may have a certain amount of atoms which were fetched from the access system, but do not belong to any molecule of the result set, because the corresponding molecule did not qualify with respect to the restricting condition. The minimization of this amount obviously is one goal of CSM optimization. We call this goal: "Detect molecule disqualification as early as possible", i.e. with as few access system calls as possible. In order to accomplish early disqualification, we must investigate the restriction clause Q. It may contain several expressions which can be evaluated of different atoms, and are combined by AND or OR. Obviously, in order to detect disqualification, one should examine the strongest condition first, i.e., the condition, which is most likely to be not fulfilled. The hierarchical structure of a molecule, however, complicates this problem. One has to decide for a starting point within the molecule structure, and then the choices where to continue are limited. Thus, the choice of a starting point also influences the time when molecule disqualification can be detected. It seems to be a good heuristic to start with an atom type scan, which is restricted by a condition. For such scans, access paths (e.g. indices) may be used. We investigate the effects of this heuristic for several shapes of Q in the following. Assume, Q is in conjunctive form, i.e. consists only of quantified terms t_i connected by "AND". We denote q(A, B, C) if an expression q can be evaluated regarding only occurrences of atom types A, B, C.

- If there is a term t_i=q(R), where R is the root atom type of the molecule, only molecules starting with atoms fulfilling t_i can be members of the result set. In this case, we can scan R restricted by t_i using the access system and then build up molecules top-down. The order of the top-down construction must be directed by the rest of the restriction clause. Those atoms, which have the highest probability to lead to molecule disqualification are fetched first. However, none of the terms still to be evaluated can be passed on to the access system because of our molecule semantics.
- If there is a term t_i of the form EXISTS q(A), where A is not the root atom type of the molecule, each molecule of the result set has to contain at least one A atom fulfilling t_i. In this case, we can start by an access system scan on A restricted by t_i, then retrieving the molecule's root atom by following the

type graph bottom up. Then, we have to build the molecule top down, knowing that it fulfils t_i. Note that the existence of a root molecule is not guaranteed. The restriction on A can be enhanced by the condition that the REFERENCE attribute pointing to the parent must not be empty, to guarantee the existence of a parent at least.

- Terms with the quantifiers EXIST_AT_LEAST(n) q(A) n≠1, EXIST_AT_MOST (n) q(A), EXIST_EXACTLY (n) q(A), and FOR_ALL q(A) are more difficult to handle. Here, one can use identifier manipulation algorithms working on the reference attributes pointing to the ancestors of those A atoms which fulfil t_i. For example, an identifier manipulation algorithm for the query

```
SELECT      ALL
   FROM     A-B
   WHERE    FOR ALL B: q(B);
```

  could work as follows: Fetch all B atoms which fulfil q(B) using an access system scan. Compute B* as the union of the identifier values of these atoms. For each A atom which is referenced by at least one of the B atoms, check whether A.b is a subset of B*. Only in this case, the corresponding molecule qualifies.

- Terms depending on more than one atom type cannot be delegated to the operations of the access system.

The top-down molecule construction builds up one molecule after the other. Thus, a pipelined processing of a CSM's result is easy to achieve. This is not true for the bottom up approach, where large sets of molecules are identified at a time.

If several of the choices are selectable, one has to find the most promising one. This decision is often driven by the existence of appropriate access paths. The problems of selectivity and access path selection are well known from relational query optimization [20]. In the case of quantified terms, the cardinality of the reference attributes also has to be taken into account. Here, a flavor of semantic query optimization appears, because the cardinality restrictions imposed in the schema declaration can be used in this context. For example, the existence of a root atom can be guaranteed, if the lower bound of all ancestor reference attributes in the corresponding path is greater than zero (cf. Example 2.2).

Things become even more complicated through the existence of atom clusters [43]. Atoms of various types are clustered according to a molecule type definition. The costs of an access to such an atom cluster is much less than the sum of the single-atom accesses. In the context of CSM, atom clusters can be used to construct whole (or parts of) molecules. This, however, may conflict with the goal of early disqualification. If the atom cluster type is not a sub-graph of the molecule type, but overlaps only partially, one has to contrast the benefits of using the atom cluster access to the cost of useless atom accesses. Some hints to the use of atom clusters are given in [43].

Cost estimations

The costs of a QEP are estimated in a similar way to the optimizer of System R [40], where a weighted sum of I/O costs and CPU utilization is computed. Nevertheless, things are little bit more complicated due to some properties of complex object processing, and in our case, due to properties of PRIMA and MAD:

- The number of atoms accessed in order to retrieve a set of molecules depends on how many atoms have to be fetched to state molecule disqualification.
- Selectivity estimations are done in analogy to [40]. Nevertheless, we have to take into account the fact that some attributes are multi-valued, and that operations like "value IN attribute value" have to be considered, e.g. for REFERENCE attributes.
- In order to estimate the number of atoms of a specific type which are involved in a query, we have to estimate the fan-out of all levels of atom types leading to this type (and have to consider the redundancy due to shared atoms therein).
- Due to the data types supported by the MAD model, which include variable length and repeating group attribute types (e.g. the REFERENCE type), atoms of the same type may strongly vary in size. Hence, in order to estimate the number of pages needed to retrieve a specific set of atoms, one has to know the average size of atoms of this type and the corresponding variance.
- Page sizes in PRIMA may vary between 0.5 and 8 kilobytes [17], but are fixed within one atom type. Furthermore, the notion of "page sequences" is supported to physically cluster a set of pages, for which I/O is more efficient than it would be for single pages. Thus, the knowledge of the number of bytes to be read is not sufficient to estimate I/O costs. Of course, basic statistic data to be held are number of pages per atom type and clustering factor. Additionally, number and average size of page sequences are to be known.
- Atom clusters are handled in a different way: here we consider the atom cluster as a whole when computing the I/O costs.

Optimizer architecture

To guarantee extensibility with respect to new optimization techniques as well as to new operators, optimization of QEPs will be performed by a rule-driven optimizer, similar to those proposed by [10, 37, 12], or better [24] which is superior with respect to evaluation simplicity. While it has been shown that such an optimizer can be generated from a given set of rules [12], the complexity of our rules is the main problem. Even in the relational case, very complex rule structures are reported [18]. The structures which have to be taken into account in our model are much more complex than those in the relational case. Hence, the language for rules used to deal with these structures also becomes more complex [14].

3.3 The Execution of Statements

The QEP of a query is an operator graph that describes an evaluation sequence at quite an abstract level. During the execution phase, the operator graph is interpreted node by node. Together with the data of the specific database it is used to compute the query's result. Obviously, there are several ways to employ concurrent computation in this process: There can be pipelining of results between parent nodes and child nodes in the operator graph (data driven execution) as well as concurrent execution of nodes in the same level of the graph. Furthermore, even concurrent computation within one operator is possible. These issues are discussed in [19].

The aspect of sharing within a result set has already been mentioned: There may be several references pointing to the same atom within a set of molecules or even within one molecule. Hence, we had to develop a representation of the resulting molecule set which does not contain multiple copies of the same data, but nevertheless allows for a separate handling of each molecule. For this purpose, we separated the representation of a molecule's structure from the representation of its data (cf. Figure 3.6). Thus, the data of an atom is not redundantly included in the result.

* * * Figure 3.6 * * *

4. Concluding Remarks

The management of complex objects occurring in advanced applications is an important new direction in current database research. Especially, efficient processing of queries on complex objects seems to be a 'hot' topic.

In this paper we addressed the problem of how to process complex-object queries. That is, basic concepts concerning query compilation, optimization and execution were investigated. A data system that transforms queries on sets of complex objects (molecules) into lower level programs for efficient execution has been introduced. Due to the dynamic derivation of complex objects from structured and heterogeneous sets of tuples, a couple of important aspects was identified concerning efficient processing:

- Our operator graph consists of nodes incorporating powerful functions; it serves as a flexible intermediate data structure that supplies optional optimization and subsequent execution.
- The efficient implementation of the hierarchical join is very important.
- The number of atom accesses, and hence the overall performance, strongly depends on a careful determination of the sequence of restriction evaluations (leading to disqualification of a molecule as early as possible).
- Depending on the restricting conditions, bottom-up or top-down construction of complex objects may be superior. In the case of bottom-up processing, a final top-down traversal may be necessary in order to complete the molecule. The choice of a strategy also influences the applicability of pipeline processing.
- Non-redundant complex-object representation requires the separation of structure from data due to the property of overlapping sub-components.
- Our concepts of enriched operator graph supports relevant extensibility measures concerning language enhancements, optimization issues, evaluation strategies, and exploitation of new access-path types (cf. [16]).

Currently, we are addressing query processing issues using our prototype system PRIMA as a test-bed for refined evaluation of these ideas and for gaining more practical experience concerning performance aspects.

In the future, special emphasis has to be placed on optimization objectives. This is of eminent importance because of the complexity of dynamic derivation and the heterogeneity of complex objects as well as the symmetric link concept and the different evaluation strategies applicable. A special aspect of optimization is the use of the parallelism inherent in complex object processing. For example, various ways of interpreting the operator graph show a different impact on the parallelism in query evaluation.

Acknowledgment

The helpful comments of W. Käfer are gratefully acknowledged.

References

[1] S. Abiteboul and N. Bidoit, Non First Normal Form Relations to Represent Hierarchically Organized Data, in: Proc. PODS Conf. (1984) 191-200.

[2] M.M. Astrahan et al., SYSTEM R: A Relational Approach to Database Management, ACM TODS 1 (1976) 97-137.

[3] D.S. Batory and A.P. Buchmann, Molecular Objects, Abstract Data Types and Data Models: A Framework, in: Proc. 10th Int. Conf. on Very Large Data Bases VLDB '84, Singapore (1984) 172-184.

[4] F. Bancilhon and S. Koshafian, A Calculus for Complex Objects, in: Proc. PODS Conf. (1986) 53-59.

[5] M.J. Carey, et al., The Architecture of the EXODUS Extensible DBMS, in: [8], 52-65.

[6] M.J. Carey, D. DeWitt, and S. Vandenberg, A Data Model and Query Language for EXODUS, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, Chicago, (1988) 413-423.

[7] G. Copeland and D. Maier, Making Smalltalk a Database System, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, Ma (1984) 316-325.

[8] K.R. Dittrich and U. Dayal (eds.), Proc. Int. Workshop on Object-Oriented Database Systems, Pacific Grove (1986).

[9] U. Dayal and N. Goodman,, Query Optimization for CODASYL Database Systems, in: ACM SIGMOD Int. Conf. on Management of Data, Orlando, Florida (1982) 138-150.

[10] K.R. Dittrich, W. Gotthard, and P.C. Lockemann, DAMOKLES - the database system for the UNIBASE software engineering environment, IEEE Data Engineering 10 (1987) 37-47.

[11] J.C. Freytag, A Rule-Based View of Query Optimization, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, San Francisco (1987) 173-180.

[12] J.C. Freytag, The Basic Principles of Query Optimization in Relational Database Management Systems, in: Proc. 11th IFIP World Computer Congress, San Francisco (1989) 801-807.

[13] G. Graefe and D.J. DeWitt, The EXODUS Optimizer Generator, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, San Francisco (1987) 160-172.

[14] R.A. Ganski and H.K.T. Wong,, Optimization of Nested SQL Queries Revisited, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, San Francisco (1987) 23-33.

[15] L. Haas, et al., Starburst Mid-Flight: As the Dust Clears, Data & Knowledge Engrg. 2 (1990) 143-160.

[16] T. Härder,, The PRIMA Project - Design and Implementation of a Non-Standard Database System, Research Report No. 26/88, SFB 124, University Kaiserslautern, 1988.

[17] L. Haas et al., Extensible Query Processing in Starburst, in: Proc ACM SIGMOD Conference on Management of Data, Portland (1989) 377-388.

[18] T. Härder et al., PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. 13th Int. Conf. on Very Large Data Bases VLDB '87, Brighton, United Kingdom (1987) 433-442.

[19] W. Hasan and H. Pirahesh, Query Rewrite Optimization in Starburst, IBM Research Report RJ6367, 1988.

[20] T. Härder, H. Schöning, and A. Sikeler, Parallelism in Processing Queries on Complex Objects, in: Jajodia, S., Kim, W., Silberschatz, A. (eds.), Proc. Int. Symp. on Databases in Parallel and Distributed Computing, Austin, Texas (1988) 131-143.

[21] M. Jarke and J. Koch, Query Optimization in Database Systems, in: Computing Surveys 16 (1984) 111-152.

[22] W. Kim, On Optimizing an SQL-like Nested Query, in: ACM TODS 7 (1982) 443-469.

[23] W. Kim,, A Model of Queries for Object-Oriented Databases, in: Proc. 15th Int. Conf. on Very Large Data Bases, VLDB '89, Amsterdam (1989) 423-432.

[24] V. Linnemann, Non First Normal Form Relations and Recursive Queries: An SQL-based Approach, in: Proc. 3rd IEEE Conf. on Data Engineering, Los Angeles (1987) 591-598.

[25] M.K. Lee, J.C. Freytag, and G.M. Lohman, Implementing an Interpreter for Functional Rules in a Query Optimizer, in: Proc. 14th Int. Conf. on Very Large Data Bases VLDB '88, Los Angeles, CA (1988) 218-229.

[26] V. Linnemann et al., Design and Implementation of an Extensible Database Management System Supporting User Defined Data Types and Functions, in: Proc. 14th Int. Conf. on Very Large Data Bases VLDB '88, Los Angeles, CA (1988) 294-305.

[27] B. Lindsay, J. McPherson, and H. Pirahesh, A Data Management Extension Architecture, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, San Francisco (1987) pp. 220-226.

[28] R.A. Lorie et al., Supporting Complex Objects in a Relational System for Engineering Databases, in: Kim, W., Reiner, D.S., Batory, D.S. (eds), Query Processing in Database Systems (Springer-Verlag, New York, 1985) 145-155.

[29] C. Lécluse, P., Richard, and F. Velez, F., O_2, an Object-Oriented Data Model, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, Chicago (1988) 424-433.

[30] Z. Manna, Z., Mathematical Theory of Computation (McGraw-Hill, 1974).

[31] B. Mitschang, Towards a unified view to design data and knowledge representation, in: Proc. 2nd Int. Conf. on Expert Database Systems, Tysons Corner VA (Benjamin/Cummings Publ. Co., 1988) 33-49.

[32] B. Mitschang, Extending the relational algebra to capture complex objects, in: Proc. 15th Int. Conf. on Very Large Data Bases, VLDB '89, Amsterdam, Netherlands (1989) 297-306.

[33] A. Meier, and R. Lorie, Implicit Hierarchical Joins for Complex Objects, IBM Research Laboratory Jan Jose, Research Report RJ3775, 1983.

[34] D. Maier et al., Development of an Object-Oriented DBMS, in: Proc. of ACM Conf. on Object-Oriented Programming Systems, Languages, and Applications (1986).

[35] K. Ono and G. Lohman, Measuring the Complexity of Join Enumeration in Relational Query Optimization, in: Proc. 15th Int. Conf. on Very Large Data Bases, VLDB '90, Brisbane, Australia (1990) 314-325.

[36] Z.M. Ozsoyoglu and L.-Y. Yuan, A Normal Form for Nested Relations, in: Proc. PODS Conf. (1985) 251-260.

[37] H.-B. Paul et al., Architecture and Implementation of the Darmstadt Database Kernel System, in: ACM SIGMOD Conf., San Francisco (1987) 196-207.

[38] A. Rosenthal and P. Helman, Understanding and Extending Transformation-Based Optimizers, in: Database Engineering 9 (1986) 44-51.

[39] M.A. Roth, H.F. Korth, and A. Silberschatz, Extended Algebra and Calculus for $\neg$ 1NF Relational Databases, Technical Report TR-84-36 of Univ. of Texas at Austin, 1985.

[40] H. Schöning, Integrating Complex Objects and Recursion, in: Proc. First Int. Conf. on Deductive and Object-Oriented Databases, Kyoto, Japan (1989) 535-554.

[41] B. Schiefer and S. Rehm, A Query Language for a Structural Object-Oriented Data Model (in German), in: T. Härder (Ed.), Database Systems in Office, Engineering, and Science, Informatik Fachberichte 204 (Springer-Verlag, Berlin, 1989) 373-388.

[42] P.G. Selinger et al., Access path Selection in a Relational Database Management System, in: ACM SIGMOD Conf, Boston (1979), 23-34.

[43] M. Stonebraker and L.A. Rowe, The Design of POSTGRES, in: Proc. ACM SIGMOD Conf., Washington, D.C. (1986) 340-355.

[44] H.-J. Schek and M.H. Scholl,, The Relational Model with Relation-Valued Attributes, in: Information Systems 2 (1986) 137-147.

[45] H. Schöning and A. Sikeler, Cluster Mechanisms Supporting the Dynamic Construction of Complex Objects, in: Proc. 3rd Int. Conf. on Foundations of Data Organization and Algorithms FODO'89, LNCS 367, Paris, France (1989) 31-46.

[46] G. Shaw and S. Zdonik, A Query Algebra for Object-Oriented Databases, in: Proc. 6th Int. Conf. on Data Engineering, Los Angeles, CA (1990) 154-162.

[47] Private communication with members of the STARBURST project at IBM Almaden Research Center, CA, 1990.

[48] SQL Addendum-2, Doc. ISO/TC97/SC21/WG3/ N143, ANSI X3 H2-86-61, 1986.

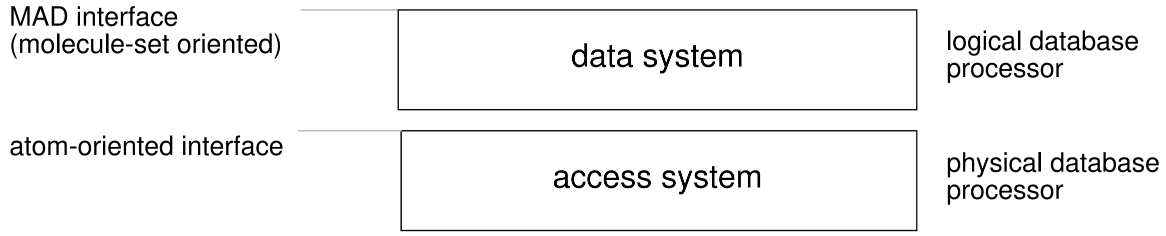


Figure 2.1: Multi-layered architecture of PRIMA

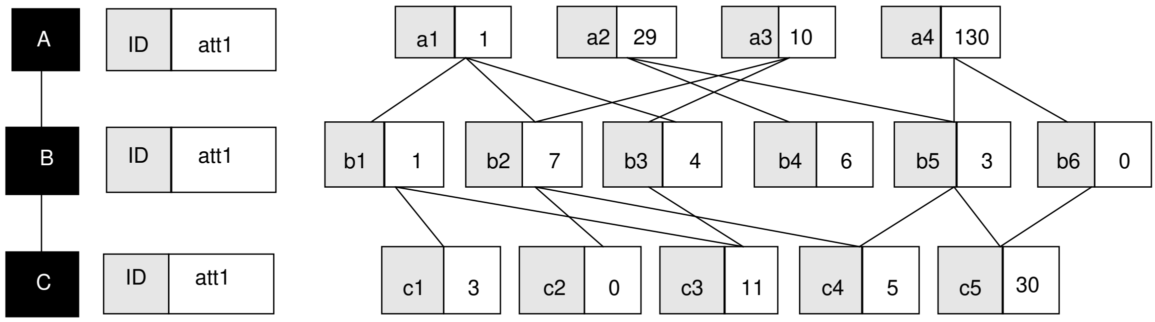
appeared in: Data and Knowledge Engineering 7, 1992, No. 3, pp. 181-200

Schema definition statements

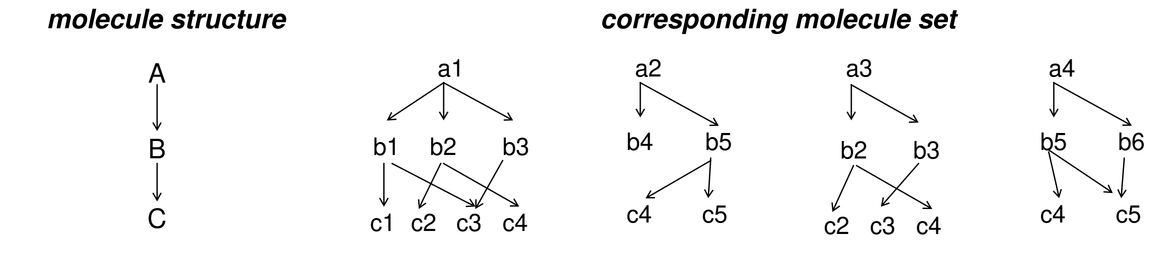
```
CREATE ATOM_TYPE A
      (ID   :  IDENTIFIER;
       att1 :  INTEGER;
       b    :  REFERENCE TO (B.a) (2,*));

CREATE ATOM_TYPE C
      (ID   :  IDENTIFIER;
       att1 :  INTEGER;
       b    :  REFERENCE TO (B.c) (1,5));

CREATE ATOM_TYPE B
      (ID   :  IDENTIFIER;
       att1 :  INTEGER;
       a    :  REFERENCE TO (A.b) (1,*);
       c    :  REFERENCE TO (C.b) (0,*));
```

resulting database schema (atom type network)

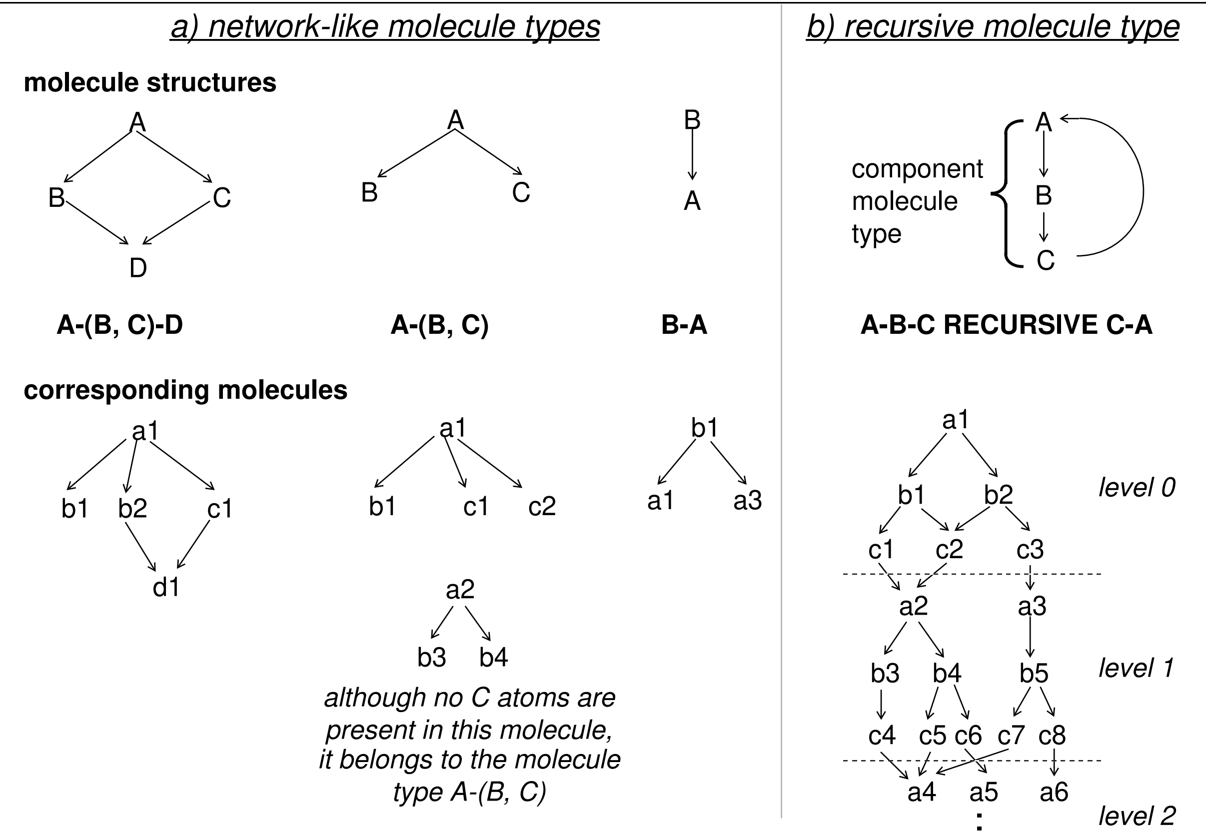
sample database (atom network)



<u>Example 2.2:</u> Sample database schema definition and sample database



Example 2.3: Sample molecule structure and corresponding molecule set based on Example 2.2



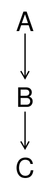
Example 2.4: Two generic kinds of molecule structures

sample query that reads as follows:

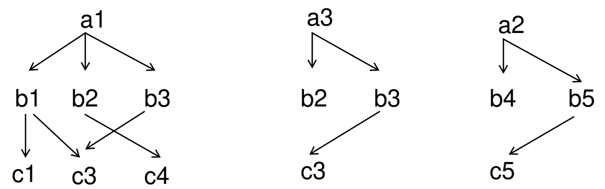
- retrieve those molecules that are constructible using A, B, and C atoms and the corresponding links, and which further satisfy the qualification B.att1>5 for at least one B atom (molecule derivation and restriction), and
- show (or project) its A atoms and all B atoms, and only those C atoms that fulfill the qualification C.att1>A.att1(qualified projection)

```
SELECT   A, B,   (SELECT C                         (* qualified projection *)
                  FROM    RESULT
                  WHERE  C.att1 > A.att1)
FROM     A-B-C                                     (* molecule-type definition *)
WHERE    EXISTS B: (B.att1 > 5);                   (* molecule-set restriction *)
```

molecule structure



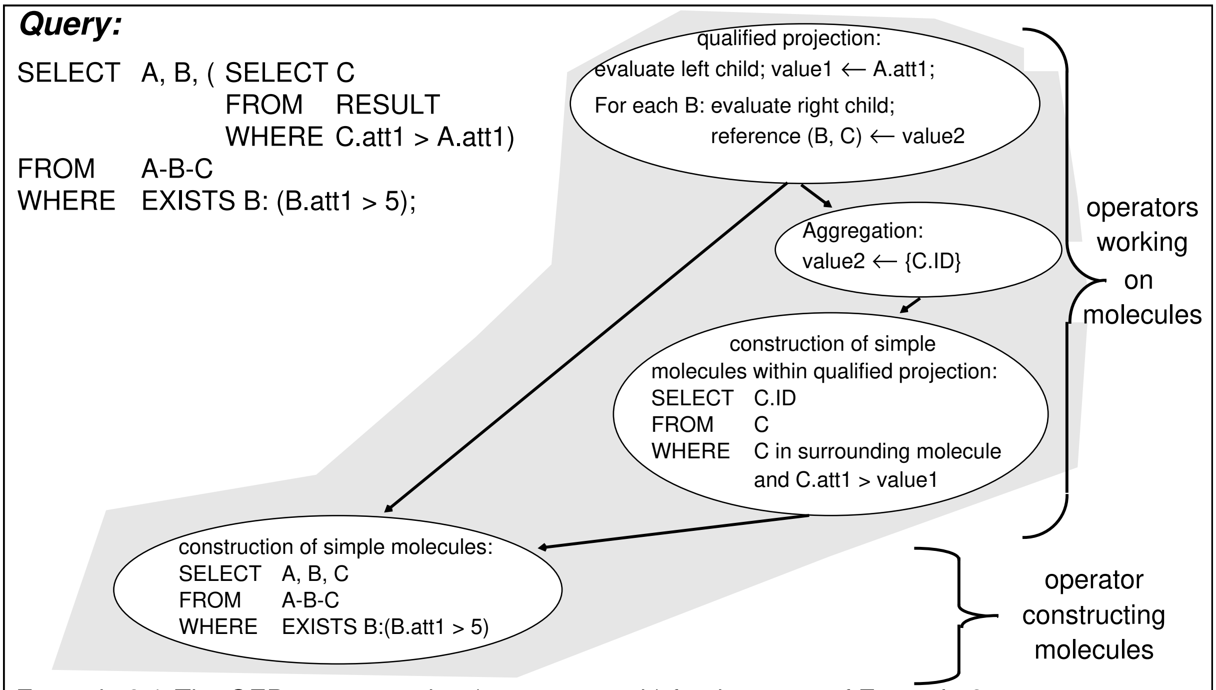
result molecule set



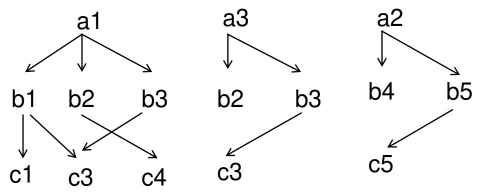
Please note the different children sets of b2 in the first and the second molecule which are the result of the qualified projection

Example 2.5: Sample query and corresponding result set based on Example 2.2

appeared in: Data and Knowledge Engineering 7, 1992, No. 3, pp. 181-200



Example 3.1: The QEP-representation (operator graph) for the query of Example 2.5



structural view to the molecule set:

each entry in the structure contains the corresponding structure building pointers as well as a pointer to the data tables shown on the right side

a1	<data>
a2	<data>
a3	<data>

b1	<data>
b2	<data>
b3	<data>
b4	<data>
b5	<data>

c1	<data>
c3	<data>
c4	<data>
c5	<data>

data tables

Figure 3.6: The two components of molecule set representation: structure and data

appeared in: Data and Knowledge Engineering 7, 1992, No. 3, pp. 181-200

Theo Haerder

Theo Haerder received the M.Sc. degree (Dipl.-Ing.) in electrical engineering and the Ph.D. degree (Dr.-Ing.) in computer science from the Technical University of Darmstadt, Germany, in 1971 and 1975, respectively.

Since 1980 he is a Professor of Computer Science at the University of Kaiserslautern, Germany, where he is the head of the Database Research Group and responsible for database education. His current research projects include the PRIMA project on advanced database systems for engineering applications, the KRISYS project on knowledge base management, and a project on high performance transaction systems. From 1977-1980 he was a Lecturer and later on a Professor of Computer Science at the Technical University of Darmstadt with a teaching and research assignment in data management topics. During the year 1976, he was a postdoctoral fellow at IBM Research in San Jose, Calif. and participated in the System R project.

Dr. Haerder is a member of ACM, ACM SIGMOD, IEEE Computer Society, IEEE TC on Data Engineering, and GI (German Computer Society). He is currently the head of the center of computer-based engineering systems at the University of Kaiserslautern.

appeared in: Data and Knowledge Engineering 7, 1992, No. 3, pp. 181-200

Bernhard Mitschang

received the M.Sc. degree (Dipl.-Inform.) and the Ph.D. degree (Dr.-Ing.) both in Computer Science from the University of Kaiserslautern, Germany, in 1982 and 1988, respectively.

Currently he manages a project in a special research program of the national science foundation (Sonderforschungsbereich 124 der Deutschen Forschungsgemeinschaft) located at the University of Kaiserslautern where he is also teaching. From October 1989 to December 1990 he was a visiting scientist in the IBM post-doctoral fellowship program at the IBM Almaden Research Center, San Jose, CA where he participated in the STARBURST project.

His research interests include various areas of database management systems, including object-oriented support, semantic modeling, query languages, query processing and optimization, and parallelism as well as application-oriented areas such as engineering information systems and knowledge base management systems.

Dr. Mitschang is a member of the Association for Computing Machinery, IEEE Computer Society, and German Computer Society (Gesellschaft für Informatik).

Harald Schöning

received the M.Sc. degree (Dipl.-Inform.) in Computer Science from the University of Kaiserslautern, Germany, in 1987. Currently he is working in a project in a special research program of the national science foundation (Sonderforschungsbereich 124 der Deutschen Forschungsgemeinschaft) located at the University of Kaiserslautern.

His research interests include query optimization in complex-object database systems, parallelism in query execution, and the mapping of advanced applications to new data models. H. Schöning is a member of the German Computer Society (Gesellschaft für Informatik).

appeared in: Data and Knowledge Engineering 7, 1992, No. 3, pp. 181-200