# Query Language for Complex Similarity Queries

Petra Budikova, Michal Batko, and Pavel Zezula

Masaryk University, Brno, Czech Republic

**Abstract.** For complex data types such as multimedia, traditional data management methods are no longer suitable. Instead of attribute matching approaches, access methods based on object similarity are becoming popular in many applications. Nowadays, efficient methods for similarity search are already available, but using them to build an actual search system still requires specialists that tune the methods and build the system. In this paper, we propose a novel query language that generalizes existing solutions and allows to formulate content-based queries in a flexible way, supporting various advanced query operations such as similarity joins, reverse nearest neighbor queries, or distinct kNN queries, as well as multi-object and multi-modal queries. The language is primarily designed to be used with the MESSIF – a framework for content-based searching – but can be employed by other retrieval systems as well.

## 1 Introduction

With the emergence of complex data types such as multimedia, traditional retrieval methods based on attribute matching are no longer satisfactory. Therefore, a new approach to searching has been proposed, exploiting the concept of similarity [17]. State-of-the-art search systems already support quite complex similarity queries with a number of features that can be adjusted according to individual user's preferences. To communicate with such a system, it is either possible to employ low-level programming tools, or a higher-level communication interface that insulates users from the implementation details. As the low-level tools can only be used by a limited number of specialists, the high-level interface becomes a necessity when common users shall be allowed to issue advanced queries or adjust the parameters of the retrieval process. In this paper, we are proposing such high-level interface in a form of a structured query language.

The motivation to study query languages arose from the development of our Metric Similarity Search Implementation Framework (MESSIF) [4]. The framework offers a wide spectrum of retrieval algorithms and is used to support several multimedia search applications, such as large-scale image search, automatic image annotation, or gait recognition. To improve the usability of our systems, we decided to offer a query language that would allow advanced users to express their preferences without having to deal with the technical details. After a thorough study of existing solutions we came to a conclusion that none of them covers all our specific needs. Therefore, we decided to propose a new language based on and extending the existing ones. At the same time, it was our desire to design the language in such a way that it could be also used by other systems.

The paper is further organized as follows. First, we review the related work in Section 2. Then we briefly summarize the requirements for a multimedia query language and present the fundamental design decisions that determined the overall structure of the language in Section 3. Section 4 introduces the theoretical model of the language, the syntax and semantics is defined in Section 5. Section 6 presents several real-world queries over multimedia data, formulated in our language. An extended version of this paper with richer related work, throughout analysis of requirements, and more examples is available as a technical report [5].

## 2 Related Work

The problem of defining a formal apparatus for similarity queries has been recognized and studied by the data processing community for more than two decades, with various research groups working on different aspects of the problem. Some of these studies focus on the underlying algebra, others deal with the query language syntax. Query languages can be further classified as SQL-based, XML-based, and others with a less common syntax. Since the algebraic operations used to express the queries are not meant to be used by users, we focus our brief survey on the query languages.

The majority of early proposals for practical query languages are based on SQL or its object-oriented alternative, OQL. Paper [10] describes MOQL, a multimedia query language based on OQL which supports spatial, temporal, and containment predicates for searching in image or video. However, similarity-based searching is unsupported in MOQL. In [8], a more flexible similarity operator for nearest neighbors is provided but its similarity measure cannot be chosen. Commercial products, such as Oracle or IBM DB2, follow the strategy outlined in the SQL/MM standard [11], which recommends to incorporate the similarity-based retrieval into SQL via user-defined data types and functions.

Much more mature extensions of relational DBMS and SQL are presented in [3, 9]. The concept of [3] enables to integrate similarity queries into SQL, using new data types with associated similarity measures and extended functionality of the select command. The authors also describe the processing of such extended SQL and discuss optimization issues. Even though the proposed SQL extension is less flexible than we need, the presented concept is sound and elaborate. The study [9] only deals with image retrieval but presents an extension of the PostgreSQL database management system that also allows to define feature extractors, create access methods and query objects by similarity. This solution is less complex than the previous one but, on the other hand, it allows users to adjust the weights of individual features for the evaluation of similarity.

Recently, we could also witness interest in XML-based languages for similarity searching. In particular, the MPEG committee has initiated a call for proposal for MPEG Query Format (MPQF). The objective is to enable easier and interoperable access to multimedia data across search engines and repositories [7]. The format supports various query types (by example, by keywords, etc.), spacio-temporal queries and queries based on user preferences. From among various proposals we may highlight [16] which presents an MPEG-7 query language that also allows to query ontologies described in OWL syntax.

Last of all, let us mention a few efforts to create easy-to-use query tools that are not based on either XML or SQL. The authors of [14] propose to issue queries via filling table skeletons and issuing weights for individual clauses, with the complex queries being realized by specifying a (visual) condition tree. Another approach [13] used the well-established Lucene query syntax.

## 3 Query Language Design

We strive to create a query language that can be used to define advanced queries over multimedia or other complex data types. The language should be general and extensible, so it can be employed with various search systems. To achieve this, we first analyzed the desired functionality of the language. Subsequently, fundamental design decisions concerning the language architecture were taken.

### 3.1 Analysis of Requirements

As detailed in [5], three sources were studied intensively to collect requirements for a multimedia query language: (1) the current trends in multimedia information retrieval, which reveal advanced features that should be supported by the language; (2) existing query languages and their philosophies, so that we can profit on previous work; and (3) the MESSIF framework architecture. The following issues were identified as the most important:

– support for a wide range of query types: in addition to various search algorithms, such as nearest neighbor search, range queries, similarity joins, sub-sequence matching, etc., single- and multi-object similarity queries as well as attribute-based (relational) and spacio-temporal queries need to be taken into consideration;
– support for multi-modal searching: multiple information sources and complex queries, combining attribute-based and similarity-based search, are a fundamental part of modern information retrieval;
– adjustability of searching: users need means of expressing their preferences in various parameter settings (e.g. precise vs. approximate search, user-defined distance functions, or distance aggregation functions);
– support for query optimization: optimizations are vital for efficient evaluation of complex queries in large-scale applications.

### 3.2 Language Fundamentals

The desired functionality of the new language comprehends the support for standard attribute-based searching which, while not being fully sufficient anymore, still remains one of the basic methods of data retrieval. A natural approach to creating a more powerful language therefore lies in extending some of the existing, well-established tools for query formulation, provided that the added functionality can be nested into it. Two advantages are gained this way: only the extended functionality needs to be defined and implemented, and the users are not forced to learn a new syntax and semantics.

The two most frequently used formalisms for attribute data querying are the relational data model with the SQL language, and the XML-based data modeling and retrieval. As we could observe in the related work, both these solutions have already been employed for multimedia searching, but they differ in their suitability for various use cases. The XML-based languages are well-suited for inter-system communication while the SQL language is more user-friendly since its query structure imitates English sentences. In addition, SQL is backed by a strong theoretical background of relational algebra, which is not in conflict with content-based data retrieval. Therefore, we decided to base our approach on the SQL language, similar to existing proposals [3, 9].

By employing the standard SQL [15] we readily gain a very complex set of functions for attribute-based retrieval but no support for similarity-based searching. Since we aim at providing a wide and extensible selection of similarity queries, it is also not possible to employ any of the existing extensions to SQL, which focus only on a few most common query operations. Therefore, we created a new enrichment of both the relational data model and the SQL syntax so that it can encompass the general content-based retrieval as discussed above.

The reasons for introducing new language primitives instead of utilizing user-defined functions are discussed in [3]. Basically, treating the content-based operations as "first-class citizens" of the language provides better opportunities for optimizations of the query evaluation. In our solution, we follow the philosophy of [3] but provide a generalized model for the content-based retrieval.

### 3.3 System Architecture

In the existing proposals for multimedia query languages based on SQL, it is always supposed that the implementing system architecture is based on RDBMS, either directly as in [9], or with the aid of a "blade" interface that filters out and processes the content-based operations while passing the regular queries to the backing database [3].

Both these solutions are valid for the query language introduced here. Since we propose to extend the SQL language by adding new language constructs, these can be easily intercepted by a "blade", evaluated by an external similarity search system, and passed back to the database where the final results are obtained. The integration into a RDBMS follows an inverse approach. The database SQL parser is updated to support the new language constructs and the similarity query is evaluated by internal operators. Of course, the actual similarity query evaluation is the cornerstone in both approaches and similarity indexes are crucial for efficient processing.

One of our priorities is creating a user-friendly tool for the MESSIF, a Java-based object-oriented library that eases the task of implementing metric similarity search systems. It offers an extensible way of defining data types and their associated metric similarity functions as well as a generic hierarchy of data manipulation and querying operations. The indexing algorithms can be plugged in as needed to efficiently evaluate different queries and the framework automatically selects indexes according to a given query. The storage backend of the MESSIF utilizes a relational database and the functionality of the standard SQL is thus internally supported. Therefore, we only need to provide a parser of the query language and a translation to native MESSIF API calls and let the framework take care of the actual execution.

## 4 Data Model and Operations

The core of any information management system is formed by data structures that store the information, and operations that allow to access and change it. To provide support for content-based retrieval, we need to revisit the data model employed by SQL and adjust it to the needs of complex data management.

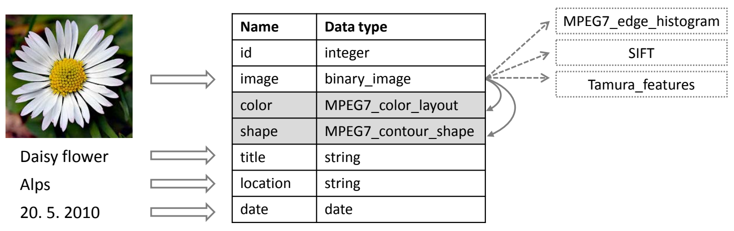
It is important to clarify here that we do not aim at defining a sophisticated algebra for content-based searching, which is being studied elsewhere. For the purpose of the language, we only need to establish the basic building blocks. Our model is based on the general framework presented in [1]. Contrary to the theoretical algebra works, we do not study the individual operations and their properties but let these be defined explicitly by the underlying search systems. However, we introduce a more fine-grained classification of objects and operations to enable their easy integration into the query language.

### 4.1 Data Model

On the concept level, multimedia objects can be analyzed using standard entity-relationship (ER) modeling. In the ER terminology, a real-world object is represented by an *entity*, formed by a set of descriptive object properties – *attributes.* The attributes need to contain all information required by target applications. In contrast to common data types used in ER modeling, which comprise mainly text and numbers, attributes of multimedia objects are often more complex (image or sound data, time series, etc.). The actual attribute values form an *n-tuple* and a set of n-tuples of the same type constitute a *relation.*

Relations and attributes (as we shall continue to call the elements of n-tuples) are the basic building blocks of the Codd's relational data model and algebra [6], upon which the SQL language is based. This model can also be employed for complex data but we need to introduce some extensions. A relation is a subset of the Cartesian product of sets $D_1$ to $D_n$, $D_i$ being the *domain* of attribute $A_i$. Standard operations over relations (selection, projection, etc.) are defined in first-order predicate logic and can be readily applied on any data, provided the predicates can be reasonably evaluated. To control this, we use the concept of *data type* that encapsulates both a an attribute domain specification and functions that can be applied on domain members. Let us note here that Codd used a similar concept of *extended data type* in [6], but he only worked with a few special properties of the data type, in particular the total ordering. As we shall discuss presently, our approach is more general. We allow for an infinite number of data types, which directly represent the primary objects (e.g. image, sound), or some derived information (e.g. color histogram). The translation of one data type into another is realized by specialized functions – *extractors*.

According to the best-practices of data modeling [15], redundant data should not be present in the relations, which also concerns derived attributes. The rationale is that the derived information requires extra storage space and introduces a threat of data inconsistency. Therefore, the derived attributes should only be computed when needed in the process of data management. In case of complex data, however, the computation (i.e. the extraction of a derived data type) can be very costly. Thus, it is more suitable to allow storing some derived attributes in relations, especially when these are used for data indexing. Naturally, more extractors may be available to derive additional attributes when asked for. Figure 1 depicts a possible representation of an image object in a relation.



| Name | Data type |
|---|---|
| id | integer |
| image | binary_image |
| color | MPEG7_color_layout |
| shape | MPEG7_contour_shape |
| title | string |
| location | string |
| date | date |

**Fig. 1.** Transformation of an image object into a relation. Full and dashed arrows on the right side depict materialized and available data type extractors, respectively.

## 4.2 Operations on Data Types

As we already stated, each data type consists of a specification of a domain of values, and a listing of available functions. As some of the functions are vital for the formulation and execution of the algebra operations, we introduce several special classes of functions that may be associated with each data type:

– *Comparison functions*: Functions of this type define total ordering of the domain ($f_C : D \times D \to \{<, =, >\}$). When a comparison function is available, standard indexing methods such as B-trees can be applied and queries using value comparison can be evaluated. Comparison functions are typically not available for multimedia data types and the data types derived from them, where no meaningful ordering of values can be defined.
– *Distance functions*: In the context of datatypes we focus on basic distance functions that evaluate the dissimilarity between two values from a given data domain ($f_D : D \times D \to \mathbb{R}_0^+$). The zero distance represents the maximal

possible similarity – identity. We do not impose any additional restrictions on the behavior of $f_D$ in general, but there exists a way of registering special properties of individual functions that will be discussed later. More than one distance function can be assigned to a data type, in that case one of the functions needs to be denoted as default. When more distance functions are available for a given data type, the preferred distance function can be specified in a relation definition. In case no distance function is provided, a trivial *identity distance* is associated to the data type, which assigns distance 0 to a pair of identical values and distance $\infty$ to any other input.
- *Extractors*: Extractor functions transform values of one data type into the values of a different data type ($f_E : D_i \rightarrow D_j$). Extractors are typically used on complex unstructured data types (such as binary image) to produce data types more suitable for indexing and retrieval (e.g. color descriptor). An arbitrary number of extractors can be associated to each data type.

In addition to the declaration of functionality, each of the mentioned operations can be equipped with a specification of various properties. The list of properties that are considered worthwhile is inherent to a particular retrieval system and depends on the data management tools employed. For instance, many indexing and retrieval techniques for similarity searching rely on certain properties of distance functions, such as the metric postulates or monotonicity. To be able to use such a technique, the system needs to ascertain that the distance function under consideration satisfies these requirements. To solve this type of inquiries in general, the set of properties that may influence the query processing is defined, and the individual functions can provide values for those properties that are relevant for the particular function. To continue with our example, the Euclidean distance will declare that it satisfies the metric postulates as well as monotonicity, while the MinimumValue distance only satisfies monotonicity. Another property worth registering is a lower-bounding relationship between two distance functions, which may be utilized during query evaluation.

### 4.3 Operations on Relations

The functionality of a search system is provided by operations that can be evaluated over relations. In addition to standard selection and join operations, multimedia search engines need to support various types of similarity-based retrieval. Due to the diversity of possible approaches to searching, we do not introduce a fixed set of operations but expect each system to maintain its own list of methods. Each operation needs to specify its input, which consists of 1) number of input relations (one for simple queries, multiple for joins), 2) expected query objects (zero, singleton, or arbitrary set), 3) operation-specific parameters, which may typically contain a specification of a distance function, distance threshold, or operation execution parameters such as approximation settings. Apart from a special case discussed later the operations return relations, typically with the scheme of the input relation or the Cartesian product of input relations. In case of similarity-based operations the scheme is enriched with additional *distance* attribute which carries the information about the actual distance of a given result object with respect to the distance function employed by the search operation.

Similar to operations on data types, operations on relations may also exhibit special properties that can be utilized with advantage by the search engine. In this case, the properties are mostly related to query optimization. As debated earlier, it is not possible to define general optimization rules for a model with a variable set of operations. However, a particular search system can maintain its own set of optimization rules together with the list of operations.

A special subset of operations on relations is formed by functions that produce scalar values. Among these, the most important are the *generalized distance* functions that operate on relations and return a single number, representing the distance of objects described by n-tuples. The input of these functions contains 1) a relation representing the object for which the distance needs to be evaluated, 2) a relation with one or more query objects, and 3) additional parameters when needed. Similar to basic distances, generalized distance functions need to be treated in a special way since their properties significantly influence the processing of a query. Depending on the architecture of the underlying search engine it may be beneficial to distinguish more types of generalized distance functions. For the MESSIF architecture in particular, we define the following two types:

- *Set distance* $f_{SD} : 2^D \times D \times (D \times D \rightarrow \mathbb{R}_0^+) \rightarrow \mathbb{R}_0^+$: The set distance function allows to evaluate the similarity of an object to a set of query objects of the same type, employing the distance function defined over the respective object type. In a typical implementation, such function may return the minimum of the distances to individual query objects.
- *Aggregated distance* $f_{AD} : (D_1 \times ... \times D_n) \times (D_1 \times ... \times D_n) \times ((D_1 \times D_1 \rightarrow \mathbb{R}_0^+) \times ... \times (D_n \times D_n \rightarrow \mathbb{R}_0^+)) \rightarrow \mathbb{R}_0^+$: The aggregation of distances is frequently employed to obtain a more complex view on object similarity. For instance, the similarity of images can be evaluated as a weighted sum of color- and shape-induced similarities. The respective weights of the partial similarities can be either fixed, or chosen by user for a specific query. Though we do not include the user-defined parameters into the definitions of the distances for easier readability, these are naturally allowed in all functions.

## 4.4 Data Indexing

While not directly related to the data model, data indexing methods are a crucial component of a retrieval system. The applicability of individual indexing techniques is limited by the properties of the target data. To be able to control the data-index compatibility or automatically choose a suitable index, the search system needs to maintain a list of available indexes and their properties. The properties can then be verified against the definition of the given data type or distance function (basic or generalized). Thus, metric index structures for similarity-based retrieval (e.g. M-tree [17], GHT* [17], M-index [12]) can only be made available for data with metric distance functions, whereas traditional

B-trees may be utilized for data domains with total ordering. It is also necessary to specify which search operations can be supported by a given index, as different data processing is needed e.g. for the nearest-neighbor and reverse-nearest-neighbor queries. Apart from the specialized indexes, any search system inherently provides the basic *Sequential Scan* algorithm as a default data access method that can support any search operation.

## 5 SimSeQL Syntax and Semantics

The SimSeQL language is designed to provide a user-friendly interface to state-of-the-art multimedia search systems. Its main contribution lies in enriching the SQL by new language constructs that enable to issue all kinds of content-based queries in a standardized manner. In accordance with the declarative paradigm of the SQL, the new constructs allow to describe the desired results while shielding users from the execution issues. On the syntactical level, the SimSeQL contributes mainly to the query formulation tools of the SQL language. Data modification and control commands are not discussed in this paper since their adaptation to the generalized data types and operations is straightforward. On the semantic level, however, the original SQL is significantly enriched by the introduction of an unlimited set of complex data types and related operations.

A SimSeQL query statement follows the same structure as SQL, being composed of the six basic clauses SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY, with their traditional semantics [15]. The extended functionality is mainly provided by a new construct SIMSEARCH, which is embedded into the FROM clause and allows to search by similarity, combine multiple sources of information, and reflect user preferences. Prior to a detailed description of the new primitives, we present the overall syntax in the following scheme:

```
SELECT    [TOP n | ALL]
          {attribute | ds.distance | ds.rank | f(params)} [, ...]
FROM      {dataset |
          SIMSEARCH [:obj [, ...]]
               IN data_source AS ds [, data_source2 [, ...]]
               BY {attribute [DISTANCE FUNCTION
                                 distance_function(params)]
                    | distance_function(params)}
               [METHOD method(params)]
WHERE     /* restrictions of attribute values */
ORDER BY  {attribute | ds.distance [, ...]}
```

In general, there are two possible approaches to incorporating primitives for content-based retrieval into the SQL syntax. We can either make the similarity search results form a new information resource on the level of other data collections in the FROM clause (an approach used in [9]), or handle the similarity as another of the conditions applied on candidate objects in the WHERE clause (exercised in [2, 3, 8, 10]). However, the latter approach requires standardized predicates for various types of similarity queries, their parameters etc., which is difficult to achieve in case an extensible set of search operations and algorithms is to be supported. In addition, the similarity predicates are of a different nature than attribute-based predicates and their efficient evaluation requires specialized data structures. Therefore, we prefer to handle similarity-based retrieval as an independent information source. Consequently, we only standardize the basic structure and expected output, which can be implemented by any number of search methods of the particular search engine.

As anticipated, the similarity-based retrieval is wrapped in the SIMSEARCH language construct, which produces a standard relation and can be seamlessly integrated into the FROM clause. The SIMSEARCH expression is composed of several parts explained in the following sections.

### 5.1 Specification of query objects

The selection of query objects follows immediately after the SIMSEARCH keyword. An arbitrary number of query objects can be provided, each object being considered an attribute that can be compared to attributes of the target relations. Multiple query objects can be used to express a more complex information need. A query object (attribute) can be represented directly by an attribute value, by a reference to an object provided externally, or by a nested query that returns the query object(s). The query objects need to be type-compatible with the attributes of the target relation they are to be compared to. Often the extractor functions can be used with advantage on the query objects.

### 5.2 Specification of a target relation

The keyword IN introduces the specification of one or more relations, elements of which are processed by the search algorithm. Naturally, each relation can be produced by a nested query.

### 5.3 Specification of a distance function

An essential part of a content-based query is the specification of a distance function. The BY subclause offers three ways of defining the distance: calling a distance function associated to an attribute, referring directly to a distance function provided by the search engine, or constructing the function within the query. In the first case, it is sufficient to enter the name of attribute to invoke its default distance function. Non-default distance function of an attribute needs to be selected via the DISTANCE FUNCTION primitive that also allows to pass additional parameters for the distance function if necessary. The last case allows greater freedom of specifying the distance function by the user, but both the attributes for which the distance is to be measured must be specified. A special function DISTANCE($x, y$) can be used to call the default distance function defined for the given data type of attributes $x, y$. The nuances of referring to a distance function can be observed in the following:

```
SIMSEARCH ... BY color
    /* search by the default distance function of the color attribute */
SIMSEARCH ... BY color DISTANCE FUNCTION color_distance
    /* search by color_distance function of the color attribute */
SIMSEARCH ... BY some_special_distance(qo, color, param)
    /* search by some_special_distance applied to the query
        object qo, color attribute, and an additional parameter */
SIMSEARCH ... BY DISTANCE(qoc, color)+DISTANCE(qos, shape)
    /* search by a user-defined sum of the default distance functions
        on qoc and qos query objects and color and shape attributes */
```

### 5.4 Specification of a search method

The final part of the SIMSEARCH construct specifies the search method or, in other words, the query type (e.g. range query, similarity join, distinct nearest neighbor search, etc.). Users may choose from a list of methods offered by the search system. It can be reasonably expected that every system supports the basic nearest neighbor query, therefore this is considered a default method in case none is specified with the METHOD keyword. The default nearest neighbor search returns all n-tuples from the target relation unless the number of nearest neighbors is specified in the SELECT clause by the TOP keyword.

The complete SIMSEARCH phrase returns a relation with a scheme of the target relation specified by the IN keyword, or the Cartesian product in case of more source relations. Moreover, information about distance of each n-tuple of the result set computed during the content-based retrieval is available. This can be used in other clauses of the query, referenced either as *distance*, when only one distance evaluation was employed, or prefixed with the named data source in the clause when ambiguity should arise (e.g. *ds.distance*).

## 6 Example Scenarios

To illustrate the wide applicability of the SimSeQL language, we now present several query examples for various use-case scenarios found in image and video retrieval. Each of them is accompanied by a short comment on the interesting language features employed. For the examples, let us suppose that the following relations, data types and functions are available in the retrieval system:

– **video_frame** relation: list of video frames

| | | |
|---|---|---|
| id | integer | identity_distance *(default)* |
| video_id | integer | identity_distance *(default)* |
| video | binary_video | identity_distance *(default)* |
| face_descriptor | number_vector | mpeg7_face_metric *(default)* |
| subtitles | string | tf_idf *(default)* |
| time_second | long | L1_metric *(default)* |

– **image** relation: register of images

| | | |
|---|---|---|
| id | integer | identity_distance *(default)* |
| image | binary_image | identity_distance *(default)* |
| color | number_vector | mpeg7_color_layout_metric *(default)*<br>L1_metric |
| shape | number_vector | mpeg7_contour_shape_metric *(default)*<br>L2_metric |
| title | string | tf_idf *(default)* |
| location | string | simple_edit_distance *(default)* |

**Query 1** *Retrieve 30 most similar images to a given example*

```
SELECT  TOP 30 id, distance
FROM    SIMSEARCH :queryImage IN image BY shape
```

This example presents the simplest possible similarity query. It employs the default nearest neighbor operation over the shape descriptor with its default distance function. User does not need any knowledge about the operations employed, only selects the means of similarity evaluation. The supplied parameter *queryImage* represents the MPEG7_contour_shape descriptor of an external query image (provided by a surrounding application). The output of the search is the list of identifiers of the most similar images with their respective distances.

**Query 2** *Find all pairs of image titles with edit distance 1 (candidates for typos)*

```
SELECT  *
FROM    SIMSEARCH
            IN image AS i1, image AS i2
            BY simple_edit_distance(i1.title, i2.title)
            METHOD MessifSimilarityJoin(1)
```

In this case, a similarity join with a threshold value 1 is required. The similarity join needs no query objects, is defined over two relations, and requires explicit reference to a distance function with the input parameters.

**Query 3** *Retrieve images most similar to a set of examples (e.g. identifying a flower by supplying several photos)*

```
SELECT  TOP 1 title
FROM    SIMSEARCH
                extract_MPEG7_color_layout(:o1) AS co1,
                extract_MPEG7_color_layout(:o2) AS co2,
                extract_MPEG7_contour_shape(:o3) AS sh3
            IN image
            BY minimum(DISTANCE(co1, color),
                DISTANCE(co2, color), DISTANCE(sh3,shape))
```

This query represents an example of a multi-object query, input of which are external binary images (denoted as *o1*, *o2*, *o3*) that are transformed to the required descriptors via extractors. Alternatively, the query objects could be provided as a result of a nested query. The minimum aggregation function employed for similarity evaluation would be formally defined on attributes and their respective distance functions. Here it is applied on the distances to individual objects only, as these are internally linked to the individual attributes and distance functions. Note that the default distance functions of the respective attributes are applied using DISTANCE$(x, y)$ construct.

**Query 4** *Retrieve all videos where Obama and Bush appear*

```
SELECT  DISTINCT vf1.video_id
FROM    SIMSEARCH :ObamaFace IN video_frame AS vf1 BY face_descriptor
            METHOD rangeQuery(0.01)
        INNER JOIN
        SIMSEARCH :BushFace IN video_frame AS vf2 BY face_descriptor
            METHOD rangeQuery(0.01)
        ON (vf1.video_id = vf2.video_id)
```

This query employs a join of two similarity search results, each of which uses a range query operation to retrieve objects very similar to the given example.

## 7 Conclusions and Future Work

In this paper, we have proposed an extensible query language for searching in complex data domains. The presented language is backed by a general model of data structures and operations, which is applicable to a wide range of search systems that offer different types of content-based functionality. Moreover, the support for data indexing and query optimization is inherently contained in the model. The SimSeQL language extends the standard SQL by new primitives that allow to formulate content-based queries in a flexible way, taking into account the functionality offered by a particular search engine. The extensibility of the presented model is achieved by the ability to define any complex data type, distance function, or similarity query operation, as well as incorporate any indexing structures that follow the design restrictions.

The proposal of the language was influenced by the MESSIF framework that offers the functionality of executing complex similarity queries on arbitrary index structures but lacks a user-friendly interface for advanced querying. Having laid the formal foundations of the query interface here, we will proceed with the implementation of a language parser which will translate the query into MESSIF for the actual evaluation. We also plan to research the possibilities of adapting the existing optimization strategies to utilize the reformulation capabilities of the proposed extension.

## Acknowledgments

This work has been partially supported by Brno Ph.D. Talent Financial Aid and by the national research projects GD 102/09/H042 and GAP 103/10/0886.

## References

1. Adali, S., Bonatti, P., Sapino, M.L., Subrahmanian, V.S.: A multi-similarity algebra. SIGMOD Rec. 27(2), 402–413 (1998)
2. Amato, G., Mainetto, G., Savino, P.: A query language for similarity-based retrieval of multimedia data. In: ADBIS. pp. 196–203. Nevsky Dialect (1997)
3. Barioni, M.C.N., Razente, H.L., Traina, A.J.M., Traina Jr., C.: Seamlessly integrating similarity queries in SQL. Softw., Pract. Exper. 39(4), 355–384 (2009)
4. Batko, M., Novak, D., Zezula, P.: MESSIF: Metric similarity search implementation framework. In: First International DELOS Conference, Revised Selected Papers. LNCS, vol. 4877, pp. 1–10. Springer (2007)
5. Budikova, P., Batko, M., Zezula, P.: Query Language for Complex Similarity Queries. Computing Research Repository (CoRR) pp. 1–22 (2012), http://arxiv.org/abs/1204.1185
6. Codd, E.F.: The relational model for database management: version 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1990)
7. Döller, M., Tous, R., Gruhne, M., Yoon, K., Sano, M., Burnett, I.S.: The MPEG Query Format: Unifying access to multimedia retrieval systems. IEEE MultiMedia 15(4), 82–95 (2008)
8. Gao, L., Wang, M., Wang, X.S., Padmanabhan, S.: Expressing and optimizing similarity-based queries in SQL. In: Conceptual Modeling - ER 2004. LNCS, vol. 3288, pp. 464–478. Springer (2004)
9. Guliato, D., de Melo, E.V., Rangayyan, R.M., Soares, R.C.: POSTGRESQL-IE: An image-handling extension for PostgreSQL. J. Digital Imaging 22(2), 149–165 (2009)
10. Li, J.Z., Özsu, M.T., Szafron, D., Oria, V.: MOQL: A multimedia object query language. In: Proc. 3rd Int. Workshop on Multimedia Information Systems (1997)
11. Melton, J., Eisenberg, A.: SQL Multimedia and Application Packages (SQL/MM). SIGMOD Record 30(4), 97–102 (2001)
12. Novak, D., Batko, M., Zezula, P.: Metric index: An efficient and scalable solution for precise and approximate similarity search. Inf. Syst. 36(4), 721–733 (2011)
13. Pein, R., Lu, J., Wolfgang, R.: An extensible query language for content based image retrieval based on Lucene. In: Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on (July 2008)
14. Schmitt, I., Schulz, N., Herstel, T.: WS-QBE: A QBE-like query language for complex multimedia queries. In: Chen, Y.P.P. (ed.) MMM. pp. 222–229. IEEE Computer Society (2005)
15. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, 6th Edition. McGraw-Hill Book Company (2011)
16. Tsinaraki, C., Christodoulakis, S.: An MPEG-7 query language and a user preference model that allow semantic retrieval and filtering of multimedia content. Multimedia Syst. 13(2), 131–153 (2007)
17. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach, Advances in Database Systems, vol. 32. Springer-Verlag (2006)