Demonstration of Index Techniques for Similarity-based Search in ORDBMSs

Michael P. Haustein[1], Wolfgang Mahnke[1], Norbert Ritter[2]

[1] Databases and Information Systems
University of Kaiserslautern
P. O. Box 3049, D-67653 Kaiserslautern
{haustein|mahnke}@informatik.uni-kl.de

[2] Distributed Systems and Information Systems
University of Hamburg
Vogt-Kölln-Straße 30, D-22527 Hamburg
ritter@informatik.uni-hamburg.de

Abstract: Today similarity-based search is used in numerous fields of applications like e-commerce, case-based reasoning, knowledge management, or text and image retrieval. To realize a similarity-based search in ORDBMSs, concepts and mechanisms are needed calculating the similarity of a comparison instance and the stored objects. Due to extremely high cost of function calls during query processing and the need to fetch all objects of the search space for calculating similarity values in order to rank the query results, it is essential to offer an index access for similarity-based queries to reduce response times.
In this demonstration, we present local indices for symbolic, numeric, and string attributes, and show the calculation of similarity values for entire objects. These indices support a new way for direct calculation of similarity values also considering table structures without having to access the actual data objects. This can lead to enormous performance benefits.

1 Introduction

Similarity-based search is deployed in numerous fields of applications. In all these applications, mechanisms are needed providing various functions to calculate the similarity of heterogeneous objects (stored in a database). Because there is no technique supporting a parametrized similarity-based access to arbitrary data in object-relational database management systems (ORDBMSs), we have to calculate a given instance's similarity to every single object stored in the database in order to get a ranked result set. Obviously, the cost for this large number of (parametrized and even nested) function calls is extremely high. Since the availability of such a complex search facility is part of the requirements of the SFB-501 Reuse Repository [MR02], we have to provide index techniques supporting a similarity-based ranked retrieval [Ha02].

2 Similarity

To calculate the similarity of two objects we first define the *distance measure* and the *similarity measure* [Be01]:

distance measure $\delta : \mathbb{D} \times \mathbb{D} \rightarrow [0, 1]$

similarity measure $sim : \mathbb{D} \times \mathbb{D} \rightarrow [0, 1]$

The distance measure δ expresses the differences of two objects in domain $\mathbb{D}$. The similarity measure *sim* calculates the similarity of two objects as a numeric value within the interval $[0, 1]$. Thereby, 0 means the least, 1 the highest similarity.

The local-global principle is used to calculate a similarity of two attribute vectors $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$, $x_i, y_i \in T_i$ and T_i is the domain of x_i resp. y_i.

First of all, we define n local similarity measures $sim_i : T_i \times T_i \rightarrow [0, 1]$ which calculate the similarity values for x_i and y_i, respectively.

Afterwards the global similarity measure $sim_\Phi(\vec{x}, \vec{y}) = \Phi(sim_1(x_1, y_1), \ldots, sim_n(x_n, y_n))$ calculates the similarity of $\vec{x}$ and $\vec{y}$ by dint of the local similarity values sim_i. The function $\Phi : [0, 1]^n \rightarrow [0, 1]$ is called the *aggregation function* and allows weighting each attribute, additionally.

As basic means for object-oriented representations, ORDBMSs [SQL99a, SQL99b] support table hierarchies and inheritance. To calculate the similarity of two objects possibly belonging to different tables within the hierarchy, we use the *intra-class* and *inter-class* similarity approach [Be01].

Thus, in a first step, the local similarities of the common attributes of the instances are calculated. Then, the local similarities are aggregated by an aggregation function to the so-called intra-class similarity. Since the structure of the table hierarchy is not considered by that method, we additionally assign a similarity value to each table. This value is supposed to express similarity in terms of the degree of structural concordance and allows the calculation of the inter-class similarity. The final similarity of two objects o_1 and o_2 is calculated as $sim_\Phi(o_1, o_2) = sim_{Intra}(o_1, o_2) \cdot sim_{Inter}(o_1, o_2)$.

3 Indices

To compute the ranked result set of a similarity query, it is inevitable to calculate the similarity of the comparison instance and each record of the queried type (possibly each record within a table hierarchy). Since every single calculation requires several expensive function calls, the query time dramatically increases with the number of records. Hence, it is mandatory to provide index structures for similarity search. Therefore, we present index tables [Ha02] which store the precalculated similarity values and allow to get ranked search results without having to calculate the similarity values at query time.

In order to precalculate the similarity of two attribute values we distinguish attribute types in symbolic, numeric, and string attributes, which covers precalculation of index data for many different data types.

To determine the similarity of two values of a symbolic attribute (an attribute which value is a reference to the actual attribute value), the similarity can be retrieved from the corresponding similarity value table. This table stores the so-called similarity matrix.

Since the domain of a symbolic attribute is finite, all potential search values are known in advance. Thus, the similarity of the attribute value and all possible search values can be computed at insert or update time of the corresponding data tuple.

Due to the infinity of numeric domains there is no conventional way of precalculating similarity values in advance of actual query evaluations. However, for similarity functions complying with some conditions [Ha02], it is possible to precalculate index data. These conditions ensure the termination of the index data calculation algorithm.

To get a similarity value for two strings we consider similarity functions using a trigram based approach. Without any additional (index) support the similarity calculations are very expensive because the functions has to operate on string data. When a data tuple is inserted, the string attribute values are decomposed to their trigrams. Each trigram is stored in a separate index table and allows an effective similarity calculation by using a group-by clause and a count function.

4 Demonstration

To demonstrate the index support for similarity based search we implemented a small Java application which performs similarity queries on a hierarchical table structure with user-defined attribute weights by use of both similarity function calls and index tables. The application sends the queries to the database system and measures the execution times. The queries are executed on a DB2 V8.1 installation on an IBM R32 Thinkpad. The database contains 100,000 data records including symbolic, numeric, and string attributes. In order to support the queries with index data, about 2,000,000 index tuples are created by triggers which guarantee integrity of the index data with regard to insert, update, and delete operations on the data tables.

The demonstration shows that similarity queries using precalculated index data can be executed in only 17 percent of the time similarity queries with conventional function calls require.

References

[Be01] Bergmann, R.: Experience Management: Foundations, Development Methodology and Internet-Based Applications, Dissertation, University of Kaiserslautern, Germany, 2001

[Ha02] Haustein, M.: Similarity Search in Object-Relational Database Systems, Diploma Thesis, Databases and Information Systems, University of Kaiserslautern, 2002, in German

[MR02] Mahnke, W.; Ritter, N.: The ORDB-based SFB-501-Reuse-Repository, Proc. 28th EDBT Conference, Software Demonstration Session, Pages 745-748, Prague, 2002

[SQL99a] American National Standard ANSI/ISO/IEC 9075-1:1999 Information Systems, Database Language - SQL Part 1: Framework, 1999

[SQL99b] American National Standard ANSI/ISO/IEC 9075-2:1999 Information Systems, Database Language - SQL Part 2: Foundation, 1999