Information Systems 59 (2016) 79–93

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

The similarity-aware relational database set operators

Wadha J. Al Marri [a], Qutaibah Malluhi [a], Mourad Ouzzani [b], Mingjie Tang [c], Walid G. Aref [c,*]

[a] Qatar University, Qatar
[b] Qatar Computing Research Institute, HBKU, Qatar
[c] Purdue University, USA

ARTICLE INFO

Available online 6 November 2015

Keywords:
Similarity query processing
Relational databases
Set operators

ABSTRACT

Identifying similarities in large datasets is an essential operation in several applications such as bioinformatics, pattern recognition, and data integration. To make a relational database management system similarity-aware, the core relational operators have to be extended. While similarity-awareness has been introduced in database engines for relational operators such as joins and group-by, little has been achieved for relational set operators, namely *Intersection*, *Difference*, and *Union*. In this paper, we propose to extend the semantics of relational set operators to take into account the similarity of values. We develop efficient query processing algorithms for evaluating them, and implement these operators inside an open-source database system, namely PostgreSQL. By extending several queries from the TPC-H benchmark to include predicates that involve similarity-based set operators, we perform extensive experiments that demonstrate up to three orders of magnitude speedup in performance over equivalent queries that only employ regular operators.

1. Introduction

Diverse applications, e.g., bioinformatics [18], data compression [36], data integration [24], and statistical classification [17], require similarity-awareness capabilities for identifying similar objects. Several similarity-aware relational operators that introduce similarity processing at the database engine level have been proposed in the past. These operators include similarity joins and similarity group-by's [28,29,26]. However, little attention has devoted to the class of relational set operations.

In standard SQL, relational set operations are based on exact matching. However, assume that we want to find common or different readings that are produced by two sensors. Assume further that the sensor readings are stored in two separate tables. The standard SQL set intersect or set difference (except) operators are not suitable for applying standard set intersection or set difference on these two sensor-data tables to get the common/different sensor readings. The reason is that sensor readings may be similar but not necessarily identical. Thus, it is desirable to perform similarity set operations on the two sensor-data tables to find similar or different readings. In this paper, we introduce similarity-aware set intersection, difference, and union as extended relational database operators.

This paper is a generalization of our previous work [16]. In addition to the similarity set intersect operator that we present in [16], in this paper, we introduce the other similarity set operators, namely similarity set difference and union. We analyze their corresponding semantics and provide efficient algorithms for each operator. In addition, we

* Corresponding author.
E-mail addresses: 200450064@student.qu.edu.qa (W.J. Al Marri), qmalluhi@qu.edu.qa (Q. Malluhi), mouzzani@qf.org.qa (M. Ouzzani), tang49@purdue.edu (M. Tang), aref@cs.purdue.edu (W.G. Aref).

http://dx.doi.org/10.1016/j.is.2015.10.008
0306-4379/

realize these operators inside an open-source relational database management system (DBMS) and provide an extensive experimental study of their performance.

The contributions of this paper are as follows:

- We introduce the similarity-aware relational set operators that extend the standard SQL relational set operators to produce results based on similarity rather than on equality (Section 3).
- We develop efficient algorithms for the proposed similarity-aware relational set operators (Section 4) and implement them inside PostgreSQL, an open-source relational database management system [20] (Section 5).
- We evaluate the performance and the scalability of the proposed algorithms using the TPC-H benchmark [33]. We extend several queries from the TPC-H benchmark by including predicates that involve similarity-based set operators. Performance results demonstrate up to three orders of magnitude enhancement in performance over equivalent queries that employ only regular relational operators (Section 5).

2. Related work

Similarity-awareness in relational operators has been mainly addressed in the relational join and group by operators. There has been work in terms of devising efficient algorithms as well integrating these similarity-aware operators inside a database engine. Another line of related research deals with nearest neighbor search as one form of similarity. In this section, we go over the main contributions in these different facets.

A nearest neighbor (NN) search finds the closest object to a query focal point. There are mainly two variants, the *k*-NN [25] and all-NN [8] operations. A *k*-NN operation identifies the *k* closest data objects to a query focal point, whereas the all-nearest-neighbor operation finds for each object in the outer table, its closest object(s) in the inner table. Lian and Chen [14] propose an efficient similarity search algorithm by employing pruning techniques to find objects in selected subspaces instead of the full space. Other performance improvement mechanisms are achieved by exploiting indexing structures, e.g., the M-tree [7] and the slim-tree [34].

Similarity join retrieves objects from the two relations that overlap based on a predefined threshold. Many types of similarity join have been proposed, e.g., [38,11,29,1,5]. *k*-nearest neighbor join (*k*-NN join) is a similarity join that combines each element in a dataset, say R, with the *k*-nearest elements in another dataset, say Böhm and Krebs [5] compute the *k*-NN join using the multipage index (MuX). MuX is an R-tree-based method to solve the optimization conflicts between the CPU cost and the I/O cost. MuX uses large-sized pages for the input data to reduce the I/O cost. Then, a secondary structure, namely buckets, with a much smaller size within pages, is used to optimize the CPU time. Recent approaches have investigated employing MapReduce to perform *k*-NN join [15] and hamming distance based similarity join [32].

The Quickjoin [13] is a metric-space algorithm that works by processing a nested-loops join on smaller subsets that are obtained after partitioning the dataset recursively. Join-Around [29] uses some properties of the distance and *k*-NN joins. In addition to having each object from the first set join with its closest object in the second set, only the pairs within a prespecified distance or radius are reported. There are several similarity join algorithms that are based on the application of grids to multidimensional datasets, e.g., Epsilon Grid Order (EGO) [4] and the Generic External Space Sweep (GESS) [9]. EGO is designed to process the similarity join on massive datasets. This solution is based on obtaining a sort order of the data objects by setting an equi-distant grid with cell length ϵ over the data space and comparing the grid cells lexicographically. GESS associates with each point an ϵ-length hypercube and then executes an intersection join on these hypercubes.

The Trie-Join [35], Fast-Join [36], ED-Join [37], Part-Enum [1], and SSjoin [6] are methods for string similarity joins. In the Trie-Join approach, a trie-based structure indexes the strings and a sub-trie pruning technique is used to efficiently perform the similarity join. SSjoin, denoting set similarity join, presents strings as sets of q-grams. Based on the string sets, SSjoin applies an overlapping function to exclude the non-matching string pairs. Then, distance computation is performed on the pairs that satisfy the overlapping condition. The other string similarity joins, namely, Part-Enum, Fast-join, and Ed-Join, employ a filter-and-refine framework. In the filter step, they produce candidate pairs by using string signatures. In the refine step, the candidate pairs are tested to see whether they are part of the final result or not.

The similarity group-by operator assigns every object to a group based on the similarity condition. A similarity-based group-by is useful in data mining applications, e.g., clustering, and duplicate detection and elimination. Group-by-Context and Group-by-Similarity are presented in [21–24,31]. Group-by-Context provides a mechanism for applying user-defined functions for grouping purposes. In contrast, Group-by-Similarity is a special case of context-aware grouping that provides the possibility to describe the similarity among tuples and grouping strategies in a descriptive way. In [28], the authors extend the standard database group-by operation to form groups of similar tuples. They implement three instances of the similarity grouping operator. Unsupervised Similarity Group-by (U-SGB) produces similarity groups based only on the specification of group properties (compactness and size). Supervised Similarity Group Around (SGB-A) forms the groups around certain central points of interest and restricts their extent based on group properties. Supervised SGB using Delimiters (SGB-D) identifies groups based on a set of delimiting points.

In order to enable similarity queries into an RDBMS, an extension to SQL to support nearest-neighbor queries is studied in [10]. This extension offers the ability to express the nearest neighbor queries in the RDBMS through a user-defined predicate termed NN-UDP. Another work [3,2] allows expressing similarity queries in SQL and executing them via a similarity retrieval engine, called SIREN (SImilarity Retrieval ENgine). SIREN is a service implemented between an RDBMS and the application programs. It processes and answers every similarity-based SQL command sent from the