SimDB: A Similarity-aware Database System

Yasin N. Silva  
Dept. of Computer Science  
Purdue University  
West Lafayette, IN, USA  
ysilva@cs.purdue.edu

Walid G. Aref  
Dept. of Computer Science  
Purdue University  
West Lafayette, IN, USA  
aref@cs.purdue.edu

Paul –A. Larson  
Microsoft Research  
Redmond, WA, USA  
palarson@microsoft.com

ABSTRACT
The identification and processing of similarities in the data play a key role in multiple application scenarios. Several types of similarity-aware operations have been studied in the literature. However, in most of the previous work, similarity-aware operations are studied in isolation from other regular or similarity-aware operations. Furthermore, most of the previous research in the area considers a standalone implementation, i.e., without any integration with a database system. In this demonstration we present SimDB, a similarity-aware database management system. SimDB supports multiple similarity-aware operations as first-class database operators. We describe the architectural changes to implement the similarity-aware operators. In particular, we present the way conventional operators’ implementation machinery is used to support similarity-aware operators. We also show how these operators interact with other similarity-aware and regular operators. In particular, we show the effectiveness of multiple equivalence rules that can be used to extend cost-based query optimization to the case of similarity-aware operations. SimDB is an open source framework that can be used by other researchers and developers to test and integrate new or improved similarity-aware operators and optimization techniques.

1. INTRODUCTION
Multiple application scenarios, e.g., marketing analysis, medical applications and data cleaning, can significantly benefit from the identification and processing of similarities in the data. Several techniques have been proposed to extend some data operations, e.g., selection and join, to process similarities in the data ([1], [2]). Unfortunately, in most of the previous work, similarity-aware operations are studied in isolation from other regular and similarity-aware operations. Furthermore, most of the previous research in the area considers a standalone implementation, i.e., without any integration with a database system. In this demonstration we present SimDB, a similarity-aware database system. SimDB supports multiple similarity-aware operations as first-class physical database operators. The implementation of these operators at the database level has the following key advantages: (1) similarity-aware operators can be interleaved with other regular or similarity-aware operators and their results pipelined for further processing; (2) important optimization techniques, e.g., pushing certain filtering operators to lower levels of the execution plan, pre-aggregation, and the use of materialized views can be extended to the new operators; and (3) the implementation of these operators can reuse and extend other operators and structures, and use the cost-based query optimizer machinery to enhance execution time. SimDB currently supports three similarity grouping and two similarity join operators. In this demonstration, we describe the architectural changes to implement the similarity-aware operators. In particular, we present the way the implementation machinery of conventional operators is extended to support similarity-aware operators. We also show practically how these operators interact with other similarity-aware and regular operators. In particular, we show experimentally the effectiveness of multiple equivalence rules that can be used to extend cost-based query optimization to similarity-aware operations. SimDB builds on the results of [3], [4], and [5].

The remaining part of the paper is organized as follows. Section 2 presents SimDB’s similarity-aware operators. Section 3 discusses the implementation of these operators and optimization techniques. Section 4 presents the demonstration scenario and Section 5 the conclusions and future work paths.

2. SimDB’s SIMILARITY-AWARE OPERATORS
The current version of SimDB supports three types of similarity grouping and two types of similarity join.

2.1 Similarity Grouping Operators
The generic definition of the similarity group-by (SGB) operator is as defined in [3]:

\[(G_1, S_1), \ldots, (G_n, S_n) \rightarrow Y_1(A_1), \ldots, Y_m(A_m) \]

where \(R\) is a relation name, \(G_i\) is an attribute of \(R\) that is used to generate the groups, i.e., a similarity grouping attribute, \(S_i\) is a segmentation of the domain of \(G_i\) in non-overlapping segments, \(F_j\) is an aggregation function, and \(A_j\) is an attribute of \(R\).

SimDB supports three instances of the previous generic definition: Unsupervised Similarity Group-by (SGB-U), Supervised Similarity Group Around (SGB-A), and Supervised SGB using Delimiters (SGB-D). SGB-U (e.g., Figure 1.a) enables grouping tuples based on desired group properties, e.g., size (MAXIMUM_GROUP_DIAMETER) and compactness (MAXIMUM_ELEMENT_SEPARATION). SGB-A (e.g., Figure 1.b) allows the grouping around points of interest. SGB-D (e.g., Figure 1.c) enables segmenting the tuples based on given limiting values. These instances represent a middle ground between the regular group-by and clustering algorithms. They are intended to be much faster than regular clustering algorithms and generate groupings that capture similarities on the data not captured by the regular group-by. As evident from Figure 1, SGB instances are able to identify successfully the naturally formed groups.
2.2 Similarity Join Operators
The generic definition of the Similarity Join (SJ) operator is as defined in [4]:

\[ A \bowtie_{\theta} B = \{(a, b) \mid \theta_{\bowtie}(a, b), a \in A, b \in B\} \]

where \( \theta_{\bowtie} \) represents the similarity join predicate. This predicate specifies the similarity-based conditions that the pairs \(<a, b>\) need to satisfy to be in the SJ output. The SJ predicates for the similarity join operators supported in SimDB are as follows.

- **Range Distance Join (F-Join):** \( \theta_{\bowtie} \equiv \text{dist}(a, b) \leq \varepsilon \)
- **Join-Around (A-Join):** \( \theta_{\bowtie} \equiv a \) is the closest neighbor of \( a \) and \( \text{dist}(a, b) \leq r \)

The \( \varepsilon \)-join operator (e.g., Figure 2.a) is an extensively used type of SJ. The Join-Around (e.g., Figure 2.b) is a useful type of SJ in which every value of the first joined set is assigned to its closest value in the second set. Additionally, only the pairs separated by a distance of at most \( r \) are part of the join output.

3. QUERY PROCESSING AND OPTIMIZATION IN SimDB

3.1 Query Processing in SimDB
SimDB extends PostgreSQL, an open source DBMS. The current implementation of similarity-aware operators in SimDB supports multiple independent numeric grouping attributes for SGB and multiple join predicates over numeric attributes for SJ.

To add support for SGB and SJ in the parser, the raw-parsing grammar rules, e.g., the \textit{yacc} rules in the case of PostgreSQL, are extended to recognize the syntax of the different new grouping approaches and join predicates. The parse-tree and query-tree data structures are extended to include the information about the type and parameters of the similarity-based operations.

In the planning stage, when multiple similarity grouping attributes (SGAs) or SJ predicates are used, they are processed one at the time. Figure 3 gives the structure of the plan trees generated when two SGAs \( a1 \) and \( a2 \) are used. The bottom aggregation node applies similarity grouping on \( a1 \) and regular aggregation on \( a2 \). The output of this node is aggregated by the top aggregation node that applies similarity grouping on \( a2 \) and regular aggregation on \( a1 \). Note that supervised aggregation nodes make use of their inner input plan tree to receive the reference points data.

Each extended aggregation node is able to process one SGA and any number of regular grouping attributes. Similarly, each extended join node can process one SJ predicate and any number of regular join predicates. The implementation of the executor routines for the SGB operators uses a single plane sweep approach to form the groups. The tuples to be grouped and the reference points data have been previously sorted and are processed simultaneously using a hash table to maintain information of the formed groups. At any time, a set of current groups is maintained and each time the sweeping plane reaches a tuple the system evaluates whether this tuple belongs to the current groups, does not belong to any group, or starts a new set of groups [3]. Range-Join and Join-Around are implemented extending the routines that support the Merge Join operator. This allows a fast and efficient implementation of both SJ operators. The sorted tuples received from the input plans are processed synchronously following also a plane sweep approach. The algorithms are coded in PostgreSQL in the fashion of a state machine. Both E-Join and Join-Around use the same set of states employed by the Sorted Merge Join.

3.2 Optimizing Similarity-aware Operators
In this demonstration, we present experimentally, how equivalence rules for similarity-aware operators can be used in SimDB to enable the transformation of queries into equivalent plans with potentially smaller expected execution time. These rules include: (1) multiple non-trivial transformation rules that exploit specific properties of SJ and SGB operators (e.g., Figure 4,[1-6]), (2) equivalence rules between multiple SJ operators and between SJ and SGB operators (e.g., Figure 4.7), and (3) Eager and Lazy aggregation transformations for SGB and SJ to enable pre-aggregation that can significantly reduce the amount of data to be processed by SJ. Figure 5 shows an example of Eager and Lazy aggregation transformation. In this case, the similarity predicate of the Join-Around (in the Lazy approach) is completely pushed down to the grouping operator (in the Eager approach). Therefore, the Eager approach avoids completely the use of the SJ operator, using instead a fast SGB operator and a regular join. In this example, the bottom grouping node of the Eager approach merges all the tuples of \( T1 \) even though they have different values of \( J1 \). Additional rules are presented in [3] and [4].
Basic Associativity of SJ Operators
1. \((E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_3} E_3 \equiv (E_1 \bowtie_{\theta_1 \land \theta_3} (E_2 \bowtie_{\theta_3} E_3))\)
2. \((E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_3} E_3 \equiv (E_1 \bowtie_{\theta_1 \lor \theta_3} (E_2 \bowtie_{\theta_3} E_3))\)

where \(\theta_1\) and \(\theta_3\) involve attributes from only \(E_1\) and \(E_2\); \(\theta_2\) and \(\theta_4\) involve attributes from only \(E_2\) and \(E_3\); \(\theta\) is a non-similarity predicate.

Associativity Rule to Enable Join on Originally Unrelated Attributes
In the case of Range Distance Join, when the attributes \(c_1\) of \(E_1\) and \(c_2\) of \(E_2\) are joined using \(\theta_1\) and \(\theta_2\) with the result \(E_3\) using \(\theta_2\), there is an implicit relationship between \(c_1\) and \(c_2\) that is exploited by the following equivalence rule:
3. \((E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2} E_3 \equiv (E_1 \bowtie_{\theta_1 \land \theta_2} (E_2 \bowtie_{\theta_2} E_3))\)

Basic Distribution of Selection over SJ
When all the attributes of the selection predicate \(\theta\) involve only the attributes of one of the expressions being joined \((E_1)\):
4. \(\sigma_{\theta}(E_1 \bowtie_{\theta_2} E_2) \equiv \sigma_{\theta}((E_1 \bowtie_{\theta_2})\bowtie_{\theta_2} E_2)\)
5. \(\sigma_{\theta}(E_1 \bowtie_{\theta_2}) E_2) \equiv \sigma_{\theta}((E_1 \bowtie_{\theta_2}) \bowtie_{\theta_2} E_2)\)

Pushing Selection Predicate under Originally Unrelated Join Operand
In equivalence rules 4-5 each selection predicate \(\theta\) is “pushed” only under the join operand that contains all the attributes referenced in \(\theta\). In the case of the Range-Join operator, the filtering benefits of pushing a selection predicate \(\theta\) can be further improved by pushing \(\theta\) under both operands of the join as shown in the following equivalence rule:
6. \(\sigma_{\theta}(E_1 \bowtie_{\theta_2} E_2) \equiv \sigma_{\theta}((E_1 \bowtie_{\theta_2}) \bowtie_{\theta_2} E_2)\)

where all the attributes of the selection predicate \(\theta\) involve only the attributes of \(E_1\), and the selection predicate \(\theta\) itself represents a modified version of \(\theta\) where each condition is “extended” by \(\theta\) and is applied on the join attribute of \(E_2\). For example, if \(0 = 10 < c_2 \leq 20\), then \(\theta = 10 - \theta < 10 - E \leq c_2 = 10 - E\).

Equivalences Among Similarity-aware Operators
Join-Around and the Similarity Group-Around are equivalent as follows:
7. \(\gamma_{\text{Join}}(E_1) = \gamma_{\text{Group}}(E_1 \bowtie_{\theta_2} E_2)\)

where \(F(\text{AA})\) is the aggregate function on aggregation attribute AA.

Figure 4. Equivalence Rules for Similarity-aware Operators

4. SimDB DEMONSTRATION SCENARIO
We will interactively show the execution and generated query plans of multiple similarity queries in SimDB. These queries make use of the different similarity-aware operators presented in Section 2. Figure 6 shows a subset of the queries to be used during the demonstration. They have been constructed extending the TPC-H benchmark [6]. For each query, we will show how SimDB computes the correct output and experimentally demonstrate how the usage of equivalence rules, like the ones presented in section 3.2, allow the generation of better execution plans.

5. CONCLUSIONS AND FUTURE WORK
We present SimDB, a similarity-aware database system that supports multiple similarity-aware operators. We describe the way these operators have been implemented and how transformation rules are used to generate better execution plans. Plans for future work include the implementation of other similarity-aware operators and the integration of indexing techniques to support similarity-aware operations at the database level.

6. REFERENCES