SimDB: A Similarity-aware Database System*

Yasin N. Silva[1] Ahmed M. Aly[1] Walid G. Aref[1] Per-Ake Larson[2]
[1]Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
{ysilva,aaly,aref}@cs.purdue.edu

[2]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 extended to support similarity-ware 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-ware operations.

Categories and Subject Descriptors
H.2.4 [**Systems**]: Query processing.

General Terms
Algorithms, Performance, Design, Experimentation.

Keywords
Similarity-aware Query Processing and Optimization, Similarity Group-by, Similarity Join.

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], [3], [4], [5], [6]). 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 multiple similarity grouping and 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-ware operations. *SimDB* builds on the results of [7], [8], and [9].
The remaining part of the paper is organized as follows. Section 2 presents the similarity-aware operators supported in *SimDB*. Section 3 discusses the implementation of these operators and several 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 several types of similarity grouping and similarity join.

2.1 Similarity Grouping Operators
The generic definition of the similarity group-by (SGB) operator is as defined in [7]:

$$_{(G_1,S_1),\ldots,(G_n,S_n)}\gamma_{F_1(A_1),\ldots,F_m(A_m)}(R)$$

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_i is an aggregation function, and A_i is an attribute of R.

SimDB supports several instances of the previous generic definition: Unsupervised Similarity Group-by (SGB-U), Supervised Similarity Group Around (SGB-A), and Supervised

* This work was partially supported by NSF Grant IIS-0811954.

SIGMOD'10, June 6–10, 2010, Indianapolis, Indiana, USA.

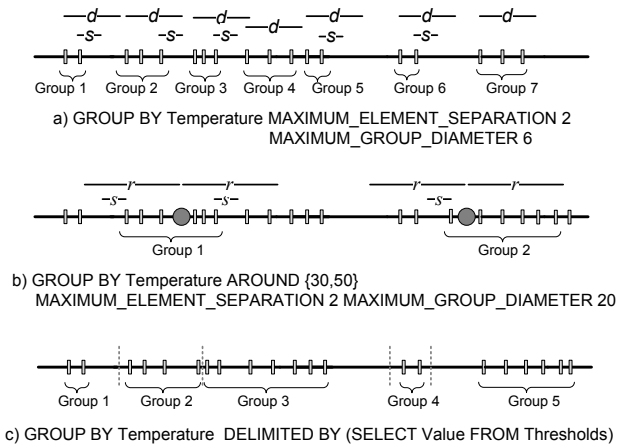

Figure 1. *SimDB*'s Similarity Group-by operators

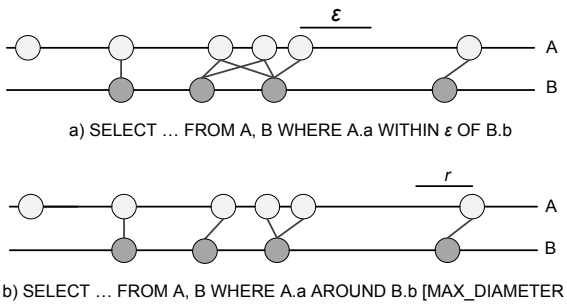

Figure 2. *SimDB*'s Similarity Join operators

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 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, similarity group-by 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 [8]:

$$A \bowtie_{\theta_S} B = \{\langle a,b\rangle \mid \theta_S(a,b), a \in A, b \in B\}$$

where θ_s 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*** *(Ɛ-Join):* $\theta_\varepsilon \equiv dist(a,b) \leq \varepsilon$
- ***Join-Around*** *(A-Join):* $\theta_{A,r} \equiv b \; is \; the \; closest \; neighbor \; of \; a \; and \; dist(a,b) \leq r$

The Ɛ-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.

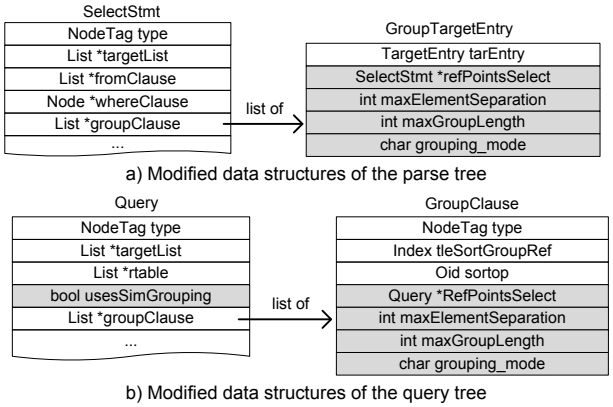

Figure 3. Modifications in the main query processing data structures to support SGB operators

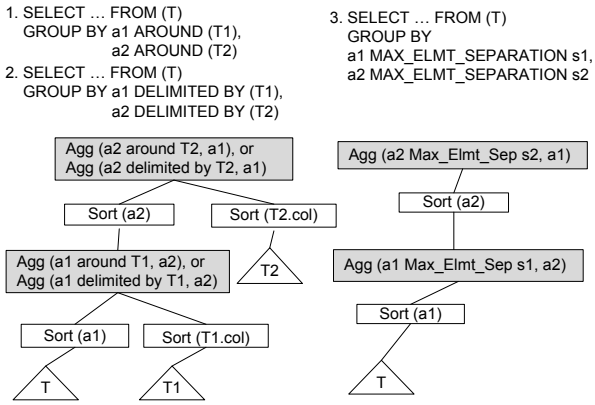

Figure 4. Path/Plan trees for SGB with multiple SGAs

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 *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. Figure 3 shows the changes in these data structures to support the SGB operators.

In the planning stage, when multiple similarity grouping attributes (SGAs) or SJ predicates are used, they are processed one at the time. Figure 4 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

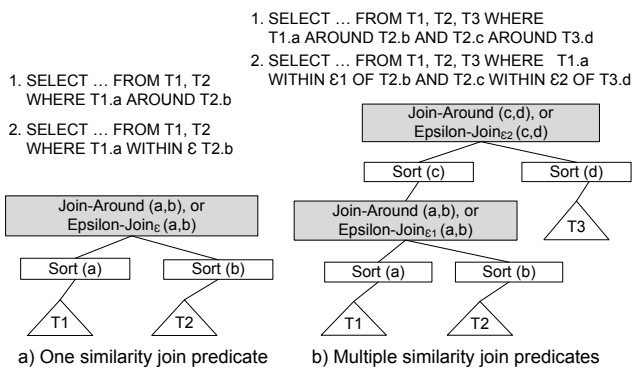

Figure 5. Path/Plan trees for SJ

Basic Associativity of SJ Operators

1. $(E_1 \bowtie_{\theta_{\varepsilon 1}} E_2) \bowtie_{\theta_{\varepsilon 2} \wedge \theta} E_3 \equiv E_1 \bowtie_{\theta_{\varepsilon 1} \wedge \theta} (E_2 \bowtie_{\theta_{\varepsilon 2}} E_3)$

2. $(E_1 \bowtie_{\theta_{A1}} E_2) \bowtie_{\theta_{A2} \wedge \theta} E_3 \equiv E_1 \bowtie_{\theta_{A1} \wedge \theta} (E_2 \bowtie_{\theta_{A2}} E_3)$

where $\theta_{\varepsilon 1}$, and θ_{A1} involve attributes from only E_1 and E_2; $\theta_{\varepsilon 2}$ and θ_{A2} involve attributes from only E_2 and E_3. θ is a non-similarity predicate.

Associativity Rule to Enable Join on Originally Unrelated Attributes

In the case of Range Distance Join, when the attributes e_1 of E_1 and e_2 of E_2 are joined using $\mathcal{E}1$ and the result joined with attribute e_3 of E_3 using $\mathcal{E}2$, there is an implicit relationship between e_1 and e_3 that is exploited by the following equivalence rule:

3. $(E_1 \bowtie_{e_1\, \theta_{\varepsilon 1}\, e_2} E_2) \bowtie_{e_2\, \theta_{\varepsilon 2}\, e_3} E_3 \equiv$
$(E_1 \bowtie_{e_1\, \theta_{\varepsilon 1 + \varepsilon 2}\, e_3} E_3) \bowtie_{(e_1\, \theta_{\varepsilon 1}\, e_2) \wedge (e_2\, \theta_{\varepsilon 2}\, e_3)} E_2$

This rule enables the generation of a cheaper plan (RHS) when the selectivity of the first join in the RHS of the rule is small.

Basic Distribution of Selection over SJ

When all the attributes of the selection predicate θ involve only the attributes of one of the expressions being joined (E_1):

4. $\sigma_\theta(E_1 \bowtie_{\theta_\varepsilon} E_2) \equiv (\sigma_\theta(E_1)) \bowtie_{\theta_\varepsilon} E_2$

5. $\sigma_\theta(E_1 \bowtie_{\theta_A} E_2) \equiv (\sigma_\theta(E_1)) \bowtie_{\theta_A} E_2$

Pushing Selection Predicate under Originally Unrelated Join Operand

In equivalence rules 4-5 each selection predicate θ is "pushed" only under the join operand that contains all the attributes referenced in θ. In the case of the Range-Join operator, the filtering benefits of pushing a selection predicate θ can be further improved by pushing θ under both operands of the join as shown in the following equivalence rule:

6. $\sigma_\theta(E_1 \bowtie_{\theta_\varepsilon} E_2) \equiv (\sigma_\theta(E_1)) \bowtie_{\theta_\varepsilon} (\sigma_{\theta \pm \varepsilon}(E_2))$

where all the attributes of the predicate θ involve only the attributes of E_1, and the selection predicate $\theta \pm Ɛ$ represents a modified version of θ where each condition is "extended" by Ɛ and is applied on the join attribute of E_2. For example, if $\theta = 10 \leq e_1 \leq 20$, then $\theta \pm Ɛ = 10 - Ɛ \leq e_2 \leq 20 + Ɛ$.

Equivalences Among Similarity-aware Operators

Join-Around and the Similarity Group-Around are equivalent as follows:

7. ${}_{e_1 \text{ around } E_2.e_2}\gamma_{F(AA)}(E_1) \equiv {}_{e_2}\gamma_{F(AA)}(E_1 \bowtie_{e_1\, \theta_A\, e_2} E_2)$

where F(AA) is the aggregate function on aggregation attribute AA.

Figure 6. Equivalence rules for similarity-aware operators

inner input plan tree to receive the reference points data. Figure 5 gives the structure of the plan trees for the cases of one and multiple SJ predicates.

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 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 [7]. Range-Join and Join-Around are implemented extending the routines that support the Sort 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 Ɛ -Join and Join-Around use the same set of states employed by the Sorted Merge Join. The main changes to implement the SJ operators are on the routine that evaluates if there is a match between two tuples and on the way the inner cursor is restored to a previous tuple to ensure the correct generation of SJ links [8].

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 6.[1-6]), (2) equivalence rules between multiple SJ operators and between SJ and SGB operators (e.g., Figure 6.7), 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 SJs. Figures 7 and 8 show examples of Eager and Lazy aggregation transformations. In figure 8, 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*. In general, both the eager and lazy versions of a query should be considered during query optimization since neither of them is the best approach in all scenarios. Joins with high selectivity tend to benefit the Lazy approach while aggregation that reduces considerably the number of tuples that flow in the pipeline tend to benefit the Eager approach. Additional rules are presented in [7] and [8].

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 9 shows a subset of the queries to be used during the demonstration. They have been constructed extending the TPC-H benchmark [10]. We will show the output generated by

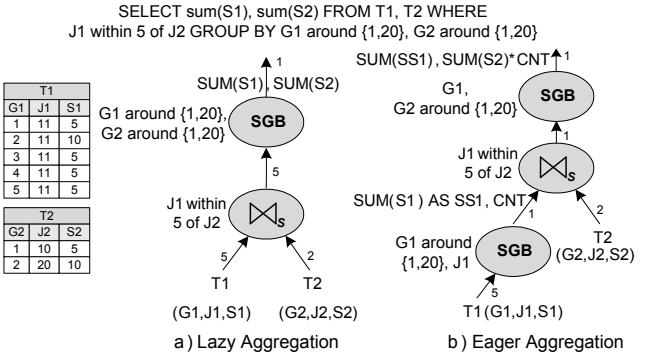

Figure 7. Eager/Lazy transformation with SGB and SJ

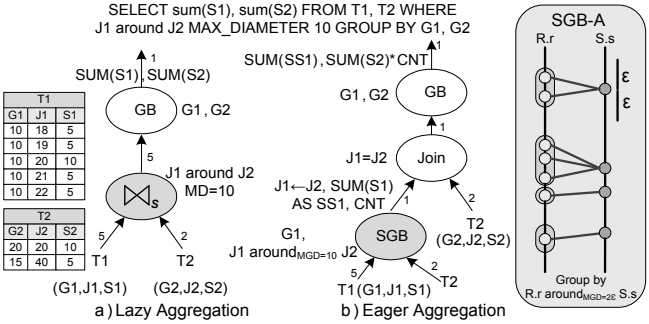

Figure 8. Pushing similarity predicate from SJ to GB

each query 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

[1] E. H. Jacox and H. Samet. Metric Space Similarity Joins. *ACM Trans. Database Syst,* 33(2): 1-38, 2008.

[2] C. Böhm, B. Braunmüller, F. Krebs, and H. P. Kriegel. Epsilon Grid Order: An Algorithm for the Similarity Join on Massive High-Dimensional Data. In *SIGMOD*, 2001.

[3] S. Chaudhuri, V. Ganti, and R. Kaushik. A Primitive Operator for Similarity Joins in Data Cleaning. In *ICDE*, 2006.

[4] X. Yang, B. Wang, and C. Li. Cost-Based Variable-Length-Gram Selection for String Collections to Support Approximate Queries Efficiently. In *SIGMOD*, 2008.

[5] Xiang Lian and Lei Chen. Similarity Search in Arbitrary Subspaces Under Lp-Norm. In *ICDE*, 2008.

[6] M. Wichterich, I. Assent, P. Kranen, and T. Seidl. Efficient EMD-based Similarity Search in Multimedia Databases via Flexible Dimensionality Reduction. In *SIGMOD*, 2008.

Similarity Query Example 1
Original TPC-H Query (Q4)
Business Question: Study how well the order priority system is working in a given quarter
Similarity-aware Query
Business Question: Study how well the order priority system works around dates of interest (holydays, marketing campaigns, etc.)
Select **d_refdate**, o_orderpriority, count(*) as order_count from orders, **DatesOfInterest** Where **o_orderdate AROUND d_refdate** and exists (Select * from lineitem Where l_orderkey = o_orderkey and l_commitdate < l_receiptdate) group by o_orderpriority, **d_refdate** order by o_orderpriority, **d_refdate**
Similarity Query Example 2
Original TPC-H Query (Q5)
Business Question: Study the revenue volume done between suppliers and customers of the same country
Similarity-aware Query
Select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue From customer, orders, lineitem, supplier, **nationSupp NS**, **nationCust NC**, region Where c_custkey = o_custkey and l_orderkey = o_orderkey and l_suppkey = s_suppkey and c_nationkey = **NC.n_nationkey** **and c_location WITHIN Ɛ TO s_location** and s_nationkey = **NS.n_nationkey** and NC.n_regionkey = NS.n_regionkey and NC.n_regionkey = r_regionkey and r_name = '[REGION]' and o_orderdate >= date '[DATE]' and o_orderdate<date '[DATE]'+interval '1' year group by n_name order by revenue desc
Similarity Query Example 3
Original TPC-H Query (Q3)
Business Question: Retrieve the unshipped orders with the highest value
Similarity-aware Query
Business Question: Clusters the unshipped orders around revenue levels of interest
SELECT **revenue as RevLevel**, count(revenue), min(revenue), max(revenue), avg (revenue) FROM (SELECT l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue FROM C, O, L WHERE c_mktsegment = 'BUILDING' and c_custkey = o_custkey and l_orderkey = o_orderkey and o_orderdate < date '1995-03-15' and l_shipdate > date '1995-03-15' GROUP BY l_orderkey) as R1 **GROUP BY revenue AROUND <RefRevLevels>**
Similarity Query Example 4
Original TPC-H Query (Q18)
Business Question: Retrieve large volume customers
Similarity-aware Query
Business Question: Retrieve clusters of customers with similar buying power
SELECT **TotalBuy as TotalBuyLevelRef**, min(TotalBuy), max(TotalBuy), count(TotalBuy), avg(TotalBuy) FROM (SELECT c_name,c_custkey,sum(l_extendedprice) as TotalBuy FROM C, O, L WHERE c_custkey = o_custkey and o_orderkey = l_orderkey and o_orderkey IN (SELECT l_orderkey FROM L GROUP BY l_orderkey HAVING sum(l_quantity) > 300) GROUP BY c_name,c_custkey) **GROUP BY TotalBuy MAXIMUM_GROUP_DIAMETER 200000** **MAXIMUM_ELEMENT_SEPARATION 20000**

Figure 9. Demonstration scenario queries

[7] Y. N. Silva, W. G. Aref, and M. H. Ali. Similarity Group-by. In *ICDE*, 2009.

[8] Y. N. Silva, W. G. Aref, and M. H. Ali. The Similarity-Join Database Operator. In *ICDE*, 2010.

[9] Y. N. Silva and W. G. Aref. Similarity-aware Query Processing and Optimization. In *VLDB PhD Workshop*, 2009.

[10] TPC-H Version 2.6.1. [Online]. Available: http://www.tpc.org/tpch.