Similarity-aware Query Processing and Optimization*

Yasin N. Silva
Supervised by: Walid G. Aref
Department of Computer Science, Purdue University
{ysilva,aref}@cs.purdue.edu

ABSTRACT

Many application scenarios, e.g., marketing analysis, sensor networks, and medical and biological applications, require or can significantly benefit from the identification and processing of similarities in the data. Even though some work has been done to extend the semantics of some operators, e.g., join and selection, to be aware of data similarities; there has not been much study on the role, interaction, and implementation of similarity-aware operators as first-class database operators. The focus of the thesis work presented in this paper is the proposal and study of several similarity-aware database operators and a systematic analysis of their role as query operators, interactions, optimizations, and implementation techniques. This paper presents the core research questions that drive our research work and the physical database operators that were studied as part of this thesis work so far, i.e., Similarity Group-by and Similarity Join. We describe multiple optimization techniques for the introduced operators. Specifically, we present: (1) multiple non-trivial equivalence rules that enable similarity query transformations, (2) Eager and Lazy aggregation transformations for Similarity Group-by and Similarity Join to allow pre-aggregation before potentially expensive joins, and (3) techniques to use materialized views to answer similarity-based queries. This paper also presents the main guidelines to implement the presented operators as integral components of a DBMS query engine and some of the key performance evaluation results of this implementation in an open source DBMS. In addition, we present the way the proposed operators are efficiently exploited to answer more useful business questions in a decision support system.

1. INTRODUCTION

It is widely recognized that the move from exact semantics of data and Boolean semantics of queries to imprecise and approximate semantics of data and queries is one of the key paradigm shifts in data management. This shift is fueled in part by the recognition that many application scenarios, e.g., marketing analysis, sensor networks, data warehousing, data cleaning, etc., require or 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., join and selection, to process data similarities. Unfortunately, there has not been much study on the role, interactions, and implementation of similarity-aware operators as first-class database operators. In this context, the research questions that drive our work are:

	Similarity Operator Implementation Approach			
	Integrated in DB Engine	Using Basic SQL Operators	Outside of DB	As Stored Procedures
Supported Operator Instances	All	Certain types may be unfeasible or require very complex queries	All	All
Implementation complexity	Can reuse and extend DB operators and structures	Queries use a complex mix of joins and aggregations	Requires specialized structures, mechanisms to deal with large data sets, etc.	Requires specialized structures, spilling mechanisms, etc.
Composable with other DB operators	Yes (full pipelining of results)	Yes (resulting queries can be highly complex)	No	No
Take advantage of DB optimizer	Yes (use of MVs, pre-aggregation, etc.)	No directly	No	No

Figure 1. Comparison of implementation approaches

1. How can DBMSs take advantage of similarities in the data to answer complex similarity-based queries required in multiple application scenarios?
2. How can standard database operators be extended to use similarities on the data?
3. How do these similarity-aware database operators interact among themselves and with the regular operators?
4. Which optimization and implementation techniques can be used to effectively realize the similarity-aware operators?

We argue that similarity-aware operators should be implemented as first-class database operators because, as shown in Figure 1, this approach has the following key advantages: (1) the similarity-aware operators can be interleaved with other regular or similarity-aware operators and its 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 to handle large datasets, and use the cost-based query optimizer machinery to enhance query execution time. Therefore, the focus of the thesis work presented in this paper is the proposal and study of several similarity-aware database operators and a systematic analysis of their role, interactions, optimizations, and implementation techniques. This paper presents the main results of the study of two key similarity-aware database operations, i.e., Similarity Group-by (SGB) and Similarity Join (SJ), proposed as part of this thesis work. The paper presents the relationships among these operators and with other regular operators. Specifically, it presents multiple equivalences rules that allow the transformation of query plans for query optimization. Key

* This work was partially supported by NSF Grant IIS-0811954 and by NIH Grant NIGMS U24 GM077905 for the EcoliHub project.

VLDB '09, August 24-28, 2009, Lyon, France.

optimization techniques proposed for regular operations are extended to the case of similarity-aware operators, e.g., Eager and Lazy aggregation transformations and using materialized views to answer queries, to the case of similarity-aware operators. The extended techniques allow for instance: (1) pushing down similarity predicates from a similarity join operator to a grouping operator, and (2) partially pushing down an aggregation or similarity aggregation operator below a join or a similarity join. The paper also presents the implementation guidelines to realize the proposed similarity-aware operators, key performance evaluation results based on their implementation in an open source DBMS (PostgeSQL), and the way the proposed similarity grouping operators are efficiently exploited to answer more useful and complex business questions in a decision support system.

The remaining part of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents the main results of the research work already conducted as part of this thesis work; specifically it presents the analysis of the Similarity Group-by and Similarity Join database operators. Section 4 presents several tasks for future work, and Section 5 the conclusions.

2. RELATED WORK

Significant work has been carried out on the extension of certain common operations, i.e., Join and Selection, to make use of similarities in the data. This work introduced the semantics of the extended operations and proposed techniques to implement them primarily as standalone operations outside of a DBMS engine rather than as integrated database operators.

Several types of similarity join and their implementation strategies, have been proposed in the literature, e.g., range distance join (retrieves all pairs whose distances are smaller than a pre-defined threshold) [1, 2, 3, 4], k-distance join (retrieves the k most-similar pairs) [5], and knn-join (retrieves, for each tuple in one table, the k nearest-neighbors in the other table) [6]. The range distance join, also known as the epsilon-join, has been the most studied type of similarity join. Among its most relevant implementation techniques, we find approaches that rely on the use of pre-built indices, e.g., eD-index [1]. These techniques strive to partition the data while clustering together similar objects. However, this approach may require rebuilding the index to support queries with different similarity parameter values, i.e., epsilon. Furthermore, eD-index is directly applicable only to the case of self-joins. Several non-index-based techniques have also been proposed to implement the range distance join. EGO [2], GESS [3], and QuickJoin [4] are three of the most relevant non-index-based algorithms. The Epsilon Grid Order (EGO) algorithm [2] imposes an epsilon-sized grid over the space and uses an efficient schedule of reads of blocks to minimize I/O. The Generic External Space Sweep (GESS) algorithm [3] creates hypersquares centered on each data point with epsilon length sides, and joins these hypersquares using a spatial join on rectangles. The Quickjoin algorithm [4] recursively partitions the data until the subsets are small enough to be efficiently processed using a nested loop join. This algorithm makes recursive calls to process each partition and a separate recursive call to process the "windows" around the partition boundary. Quickjoin outperforms EGO and GESS [4]. Also of importance, is the work on similarity join techniques that make use of relational database technology, e.g., [7]. These techniques are applicable only to string or set-based data. The general approach pre-processes the data and query, e.g., decomposes data and query strings into sets of q-grams, and stores the results of this stage on separate tables. Then, the result of the similarity join can be obtained using standard aggregate/group-by/join SQL statements. Indices on the pre-processed data are used to improve performance.

The special cases of similarity joins with one-tuple inner relations represent several types of similarity selection. Among key recent contributions on similarity selection we have: a quantitative cost-based approach to build high-quality grams to support selection queries on strings [8], a method to find all data objects that match with a given query object in a low-dimensional subspace instead of the original full space [9], and flexible dimensionality reduction techniques to support similarity search [10].

The work in [11] proposes an algebra for similarity-based queries. This work presents the extension of simple algebra rules, e.g., pushing selection into join, to the case of similarity operators. A framework for similarity query optimization is presented in [12]. This work makes use of simple equivalence rules to generate alternative query plans. The main difference between this body of work and our contribution is that we focus on analyzing in detail the properties among different types of similarity-aware operators, among different instances of the same similarity operator, and among regular and similarity-aware operators. Furthermore, we study the extension of query optimization techniques, e.g., lazy and eager aggregation transformations, and the use of materialized views to answer queries, to the case of similarity-based queries.

The work on clustering techniques developed in various fields, e.g., pattern recognition, machine learning, biology; represents also a related area which studies ways to group together similar objects. Of special interest is the work on clustering of very large datasets. CURE [13] and BIRCH [14] are two of the most representative clustering algorithms. They are based on sampling and summaries, respectively. CURE and BIRCH use only one pass over the data and hence reduce notably the execution time for clustering. However, when compared to the execution time of the standard group-by operation, the execution time of CURE and BIRCH are significantly slower. Furthermore, the use of clustering is via a complex data mining model and its implementation is not integrated with the standard query processing engine.

In the context of data reconciliation, Schallehn et al. propose SQL extensions to allow the use of user-defined similarity functions for grouping purposes [15] and similarity grouping predicates [16]. They focus on string similarity and similarity predicates to reconcile records. Although they can be used for this purpose, the proposed similarity group-by operators in this paper are more general and are designed to be part of a DBMS's query engine.

Some of the optimization techniques of similarity join presented in this paper build on previous work on optimization of regular non similarity queries. Larson et al. study pull-up and push-down techniques that allow the query optimizer to move aggregation operators up and down the query plan [17, 18]. These techniques enable complete [17] or partial [18] pre-aggregation that can reduce significantly the input size of a join and decrease the execution time of an aggregation query. The use of materialized views to answer aggregation queries is another technique that can dramatically improve the execution time of certain queries [19].

3. OUR RESEARCH CONTRIBUTIONS

This section presents briefly the main results of the research work already conducted as part of this thesis work [20, 21, 22].

3.1 Similarity-aware Operators

This subsection presents the similarity-aware counterparts of two core database operations: Group-by and Join.

3.1.1 Similarity Group-by

The generic definition of the similarity group-by (SGB) operator is as follows [20]:

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

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.

In addition we introduce three implementable 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 2.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 2.b) allows the grouping around points of interest. SGB-D (e.g., Figure 2.c) enables segmenting the tuples based on given limiting values. Several instances can be combined in the same query [20]. These instances represent a middle ground between the regular group-by and standard 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 regular group-by. As evident from Figure 2, SGB instances are able to identify successfully the naturally formed groups.

3.1.2 Similarity Join

The generic definition of the Similarity Join (SJ) operator is as follows [22]:

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

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 considered in our study are as follows.

- ***Range Distance Join*** *(Ɛ-Join):* $\theta_\varepsilon \equiv dist(a,b) \leq \varepsilon$
- ***kNN-Join****:* $\theta_{kNN} \equiv b \; is \; a \; k-closest \; neighbor \; of \; a$
- ***k-Distance-Join*** *(kD-Join):* $\theta_{kD} \equiv \langle a,b\rangle \; is \; one \; of \; the \; overall \; k-closest \; pairs$
- ***Join-Around*** *(A-Join):* $\theta_{A,MD=2r} \equiv b \; is \; the \; closest \; neighbor \; of \; a \; and \; dist(a,b) \leq r$

The Ɛ-, kNN-, and kD-join operators are common and extensively used types of similarity join. The Join-Around is a new useful type of similarity join that combines some properties of both the range distance and kNN joins. 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. Here *MD* stands for *Maximum Diameter* and *r=MD*/2 *represents the Maximum Radius*. Figure 3 shows the extended SQL and examples of the four types of similarity join operators.

3.2 Optimizing Similarity-aware Operators

Multiple optimization techniques are studied for the proposed similarity-aware database operators. These optimization techniques are presented in detail in [20] and [22] and include: (1) multiple non-trivial transformation rules that exploit specific properties of SJ and SGB operators, (2) equivalence rules between

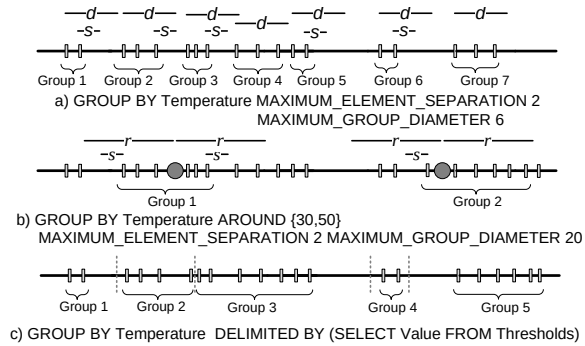

Figure 2. Examples of Similarity Group-by

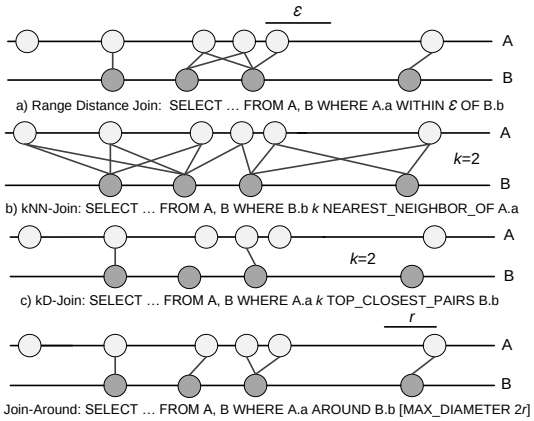

Figure 3. Examples of Similarity Join

multiple SJ operators and between SJ and SGB operators, (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, and (4) techniques to use materialized views to answer similarity-based queries.

Equivalence rules enable the transformation of queries into equivalent plans with potentially smaller expected execution time. We propose multiple non-trivial equivalence rules for introduced similarity-aware operators. Figure 4 presents a subset of them. Additional rules are presented in [22] and [20].

Another important query optimization approach is the use of pull-up and push-down techniques to move the grouping operator up and down the query tree. The main Eager and Lazy aggregations theorem introduced in [17] enables several pull-up and push-down techniques for the regular, i.e., non-similarity, join and group-by operators. We have extended the main theorem to the cases of: (1) regular join and similarity group-by, (2) similarity join and regular group-by, and (3) similarity join and similarity group by. Figures 5 and 6 illustrate the first two cases respectively. In general, the single aggregation operator of the Lazy approach is split into two parts in the Eager approach. The first part pre-evaluates some aggregation functions and calculates the count before the join. The second part uses the intermediate information to calculate the final results after the join. 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. Moreover, we study

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

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

where $\theta_{\varepsilon 1}$, θ_{A1}, and θ_{kNN1} involve attributes from only E_1 and E_2; $\theta_{\varepsilon 2}$, θ_{A2}, and θ_{kNN2} involve attributes from only E_2 and E_3. θ is a non-similarity predicate. [22] presents more rules that combine different types of SJ.

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:

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

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):

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

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

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

Pushing Selection Predicate under Originally Unrelated Join Operand

In equivalence rules 5-7 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:

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

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

Equivalences Among Similarity-aware Operators

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

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

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

Join-Around and kNN-Join are equivalent under the following rule:

10. $E_1 \bowtie_{\theta_{A,MD=\infty}} E_2 \equiv E_1 \bowtie_{\theta_{kNN\ (k=1)}} E_2$

Further equivalences among different SJ operators are presented in [22].

Figure 4. Equivalence Rules for Similarity-aware Operators

ways to further enhance the filtering power of the pre-aggregation step pushing down the similarity predicates from the SJ operator to the grouping one. Specifically we propose extensions that support (1) pushing similarity predicates from Range-Join to GB and (2) pushing similarity predicate from Join-Around to GB [22].

Using materialized views to answer queries [19] is another important optimization technique that can yield considerable query processing time improvements and can be extended to the case of similarity-aware operators. View matching algorithms to determine if a certain query can be answered using a set of materialized views are presented in [19]. These algorithms are extended to the case of queries and views with SGB in [20] and to the case of queries and views with SJ in [22].

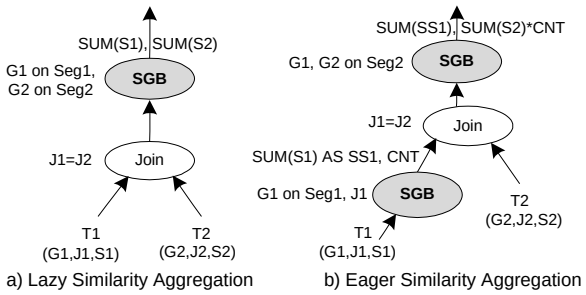

Figure 5. Eager/Lazy Transformation with SGB and Join

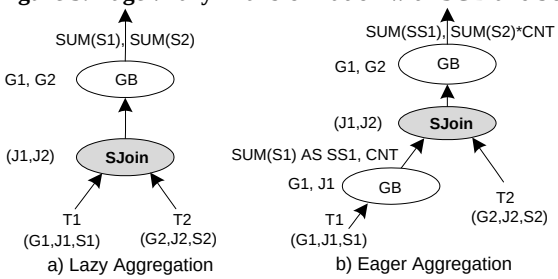

Figure 6. Eager/Lazy Transformation with GB and SJ

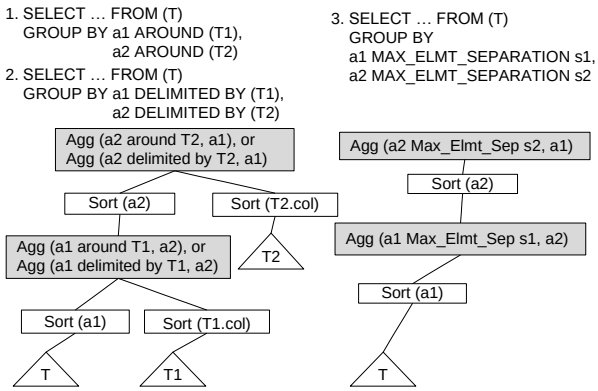

Figure 7. Path/Plan Trees for SGB with multiple SGAs

3.3 Implementing Similarity-aware Operators

The detailed algorithms and data structures to implement all three instances of SGB as well as the Range-Join and Join-Around inside the query engine of standard RDBMSs are presented in [20] and [22]. This subsection briefly presents the main implementation guidelines. This implementation considers the support of multiple independent numeric grouping attributes for SGB and multiple join predicates over numeric attributes for SJ. One of the goals of the implementation is to reuse and extend already available routines and structures to minimize the effort needed to realize these operators.

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.

In the planning stage, when multiple similarity grouping attributes (SGAs) or SJ predicates are used, they are processed one at the time. Figure 7 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.

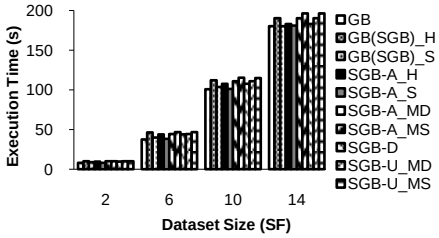

Figure 8. Performance of SGB while increasing dataset size

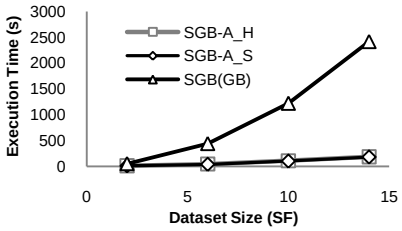

Figure 9. Performance of generating similarity groups with GB vs. SGB

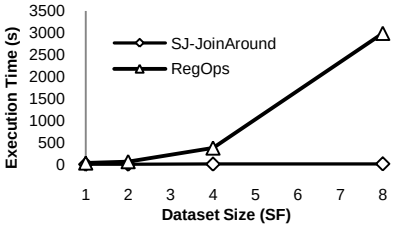

Figure 10. Performance of Join-Around (Nearest Neighbor Join)

The output of this node is aggregated by the top aggregation node that applies similarity grouping on a2 and regular aggregation on a1. Notice 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 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. 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.

3.4 Performance Evaluation

The proposed similarity-aware operators were implemented in an open source database (PostgreSQL), and their performance is studied in detail in [20] and [22]. This subsection presents some of the key results. The dataset used in the performance evaluation is based on the one specified by the TPC-H benchmark. Figure 8 gives the execution time of several aggregation queries for different dataset sizes. The key result of this experiment is that the execution times of all the queries that use similarity group-by, i.e., SGB-X, are very close to the execution time of the regular aggregation query GB for all the dataset sizes. Even in the worst case scenario represented by GB(SGB)_X, i.e., SGB query produces the same result as GB, the execution time of GB(SGB) is at most only 25% bigger than the one of GB. Although in general it is not possible to produce the output of SGB queries using only regular SQL operations, this is feasible in some special cases, e.g., SGB-A without additional conditions. Figure 9 compares the execution time of SGB(GB) with that of SGB-A. The presented results show that the execution time and scalability properties of the SGB query is much better than those of the query that uses only regular SQL operations. The execution time of SGB(GB) grows from being 500% bigger than that of SGB-A for SF(scale factor)=1 to being 1300% bigger for SF=14. Figure 10 gives the execution time of the Join-Around query SJ-JoinAround compared to RegOps query that produces the same output using only regular database operators. For all analyzed SF values, the Join-Around query significantly outperforms the non-similarity-based query. The execution time of SJ-RegOps grows from being about 20 times bigger than that of JoinAround for SF=1 to being about 200 times bigger for SF=8.

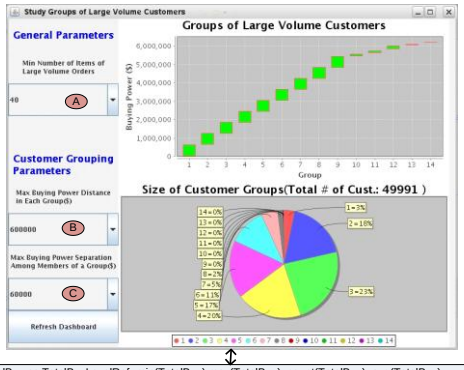

Figure 11. Exploiting SGB to identify clusters of large-volume customers with similar buying power

3.5 Exploiting Similarity-aware Operators

The proposed similarity-aware operators could be used in many application scenarios. In [21], we study the way the similarity grouping operators can be exploited to answer interesting business questions in a decision support system (DSS). Two key properties of SGB that are of special importance for this application are: the fast formation of groups which allows the construction of dynamic dashboards, and the pipelining of results which allows further processing of the SGB results. One of the DSS dashboards is shown in Figure 11. This dashboard allows the study of groups of customers with similar buying power, i.e., total revenue due to large volume orders. The dashboard allows the dynamic specification of properties that describe the desired similarity groups: group-size (maximum buying power distance in each group), and compactness (maximum buying power separation among members of a group). Other dashboards support the study of profit around marketing campaigns, and the analysis of groups of orders around revenue levels of interest [21].

4. FUTURE WORK

This section presents the future tasks of the presented thesis work.

Other similarity-aware database operators. Our previous work focused on the SGB and SJ operators. We plan on studying

the similarity-aware counterparts of other database operators, e.g., selection, duplicate elimination, set intersection, and set difference. Similarly to the work carried out for SGB and SJ, we plan on studying optimization techniques for these operators, e.g., we expect to identify new equivalence rules over these operators.

*Similarity-aware data warehousing operators***.** The CUBE and ROLLUP operators, which are extensively used in data warehousing applications, can be extended to use similarity grouping mechanisms like the ones used in SGB. Different similarity grouping strategies can be used to group the values in different dimensions. These extended CUBE and ROLLUP operators will be able to generate more meaningful and useful summaries of large datasets.

*Benchmark for Similarity-based Query Processing***.** We plan on building a benchmark to evaluate the similarity-aware query processing capabilities of database systems. This benchmark is expected to extend the TPC-H benchmark with data and queries that allow answering multiple complex business questions. The queries are expected to exploit similarities in the data and have broad industry-wide relevance.

*Further practical application of the proposed similarity-aware operators***.** We intend to exploit all the proposed similarity-aware operators to enable more useful and complex business analysis extending the DSS presented in Section 3.5. Furthermore, we plan on studying the use of similarity-aware operators in the problem of phenomena detection in sensor networks, where phenomenon can be defined as a persistent condition observed in a set of sensors. Similarity-based queries are especially useful in sensor data processing given the approximate nature of sensor data. Similarity-based queries can be actively used to specify and detect phenomena.

5. CONCLUSIONS

Many application scenarios need or can benefit tremendously from database operators that exploit similarities in the data and allow the pipelining of the results for further processing. Related previous work has focused on the extension of the semantics of some operations and has proposed mainly standalone implementation techniques that are not fully integrated with the query processing engine of DBMSs. The focus of this thesis work is the proposal and study of several similarity-aware database operators and the analysis of their role as query operators, interactions, optimizations, and implementation techniques. This paper presents the main results for the operators studied so far: Similarity Group-by and Similarity Join. This presentation includes: (1) the generic definition and several instances of each operator, (2) multiple optimization techniques for the introduced operators, (3) guidelines to implement them as integral components of a DBMS query engine, (4) performance evaluation results, and (5) the practical usage of the proposed similarity grouping operators to answer more useful and complex business questions in a decision support system. The paper describes also the future tasks of the presented thesis work.

6. ACKNOWLEDGMENTS

Our thanks to Mohamed H. Ali and Muhammad U. Arshad for their valuable ideas and help to realize the SGB-based DSS.

7. REFERENCES

[1] V. Dohnal, C. Gennaro, and P. Zezula. Similarity Join in Metric Spaces Using eD-Index. In *DEXA*, 2003.

[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] J. P. Dittrich and B. Seeger. GESS: a Scalable SimilarityJoin Algorithm for Mining Large Data Sets in High Dimensional Spaces. In *SIGKDD* 2001.

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

[5] G. Hjaltason, and H. Samet. Incremental distance join algorithms for spatial databases. In *SIGMOD*, 1998.

[6] C. Böhm and F. Krebs. The k-Nearest Neighbour Join: Turbo charging the KDD process. *Knowledge and Information Systems*, 6(6): 728-749, 2004.

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

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

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

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

[11] S. Adali, P. Bonatti, M. L. Sapino, and V. S. Subrahmanian. A Multi-Similarity Algebra. In *SIGMOD*, 1998.

[12] M. R. Ferreira, C. Traina, and A. J. Traina. An Efficient Framework for Similarity Query Optimization. In *GIS*, 2007.

[13] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large datasets. In *SIGMOD*, 1998.

[14] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25(2): 103-114, 1996.

[15] E. Schallehn, K. Sattler, and G. Saake. Extensible Grouping and Aggregation for Data Reconciliation. In *EFIS*, 2001.

[16] E. Schallehn, K. Sattler, and G. Saake. Efficient similarity-based operations for data integration. *Data & Knowledge Engineering*, 48(3): 361-387, 2004.

[17] W. Yan and P. Larson. Eager Aggregation and Lazy Aggregation. In *VLDB*, 1995.

[18] P. Larson. Data reduction by partial preaggregation. In *ICDE*, 2002.

[19] J. Goldstein and P. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *SIGMOD*, 2001.

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

[21] Y. N. Silva, M. U. Arshad, and W. G. Aref. Exploiting Similarity-aware Grouping in Decision Support Systems. In *EDBT*, 2009.

[22] Y. N. Silva, W. G. Aref, and Mohamed H. Ali. Similarity Join Database Operators. Technical Report, Department of Computer Science, Purdue University, 2009.