Efficient Semantically Equal Join on Strings in Practice

Juggapong Natwichai
Computer Engineering Department, Faculty of Engineering
Chiang Mai University, Chiang Mai, Thailand 50200
juggapong@chiangmai.ac.th

Xingzhi Sun
IBM Research Laboratory, Beijing, China
sunxingz@cn.ibm.com

Maria E. Orlowska
School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, Qld 4072, Australia
maria@itee.uq.edu.au

Abstract

In general, data integration begins with schema integration and must be followed by detailed data instances resemblance in order to reach data representation unification. In this paper, we address the limits of the data-level reconciliation automation process in cases where the compared data is semantically equivalent, but the data representation of the values of given attributes is different. We assume that a semantic relationship between potentially used terms is established by a human expert prior to the designed computations, and is represented in an auxiliary table built and maintained for each attribute.

On such a context, we introduce a notion of *semantically equal join (SEJ),* which is the join operation based on a pre-defined semantic relationship. Our goal is to propose a solution for SEJ that can be supported by standard SQL. The paper begins with an illustration of the approach for a single attribute join, followed by a generalisation of SEJ for any number of join attributes. The paper continues with performance considerations for the invented method. Finally, this aspect is supported by extensive experimentation based on our implementation of SEJ executed on synthetic datasets.

Keywords: Approximate String Join, Data Integration, Semantically Equal Join.

1. Introduction

The data integration issue [**1**] is one of the most challenging problems that computer science and IT practitioners have faced in the last decade or so. This problem originally emerged when demand for data access from multiple sources was observed in the context of distributed multi-database systems; later, database federation systems and other versions of data integrated constructions including data warehousing and, very recently, e-research applications.

Firstly, integration problems occur at the schema level [**2**] where the name, meaning and constraint scope of attributes must be addressed. Usually, human involvement in this process is necessary [**3**]. Subsequently, the data-level integration problems need to be addressed. The problem at this level exists due to: potential mismatch of attributes' domains, adopted strings to represent attributes' values, as well as conventions to express the data fields. Additionally, the data mismatch may be caused by many other reasons, such as, for instance, typing errors.

In this paper, we consider the data integration where two given relations are joined with each other on some attributes. On this context, we address the *synonym mismatch problem*, i.e., the values of join attributes are semantically equivalent, but unable to match due to different representation, e.g. different abbreviation standards. Naturally, on such an attribute domain, the *semantic equivalence relationship* between strings needs to be pre-defined. We call the join operations that are based on this semantic equivalence relationship, *semantically equal join (SEJ)*.

For example, from Figure 1a, R_1 and R_2 are two relations which contain the medical records of two different hospitals respectively. Assume that one issues a query to find pairs of patients who have the same type of disease. Due to the fact that in medical domain, strings "Tumours", "Kunb", "Neoplasms" are semantically equivalent and so are the strings "Mad Cow" and "B.S.E." (Bovine Spongiform Encephalopathy), the result of the query should be the relation given in Figure 1b. We can see that this query result can be obtained by joining two relations R_1 and R_2 on their common attribute "Disease" such that the join condition is true if and only if the values of the join attribute are semantically equivalent.

In the rest of this paper, the focus is to formally define the SEJ operation and to propose an efficient approach to compute SEJ in a standard SQL environment.

Our discussion begins with an illustration of the approach for 1-attribute SEJ. First, to represent the semantic relationship between potentially used terms, we introduce the *auxiliary table* which is an additional relation built and maintained for the pair of attributes on which the join is to be performed. The design of the auxiliary table is important because it directly affects the SEJ performance. To address this issue, we design the schema of the auxiliary table, based on which, the relational algebra for SEJ is proposed.

Since, in general, the way that the SQL statement is written will affect the performance of queries in RDBMS, we optimize the query performance for the group-based approach by adjusting its SQL formulation. Particularly, we propose a sub-query-based SQL statement for SEJ, which is much more efficient than the SQL, that is directly translated from the relational algebra.

PID	**Disease**
P_1	Tumours
P_2	Knub
P_3	Mad Cow
P_4	Bird Flu

R_1

PID	**Disease**
Q_1	Neoplasms
Q_2	B. S. E
Q_3	Bird Flu

R_2

a) Example relation

PID	**Disease**	**PID**	**Disease**
P_1	Tumours	Q_1	Neoplasms
P_2	Knub	Q_1	Neoplasms
P_3	Mad Cow	Q_2	B. S. E
P_4	Bird Flu	Q_3	Bird Flu

R_1 R_2

b) SEJ result

Figure 1: SEJ Example

Learning from discussion of 1-attribute SEJ, we generalize the SEJ problem to *k*-attributes, where *k* auxiliary tables need to be designed and maintained. Based on the sub-query-based SQL statement proposed for 1-attribute SEJ, it is straightforward to get a query for computing *k*-attributes SEJ. Because the join condition of *k*-attributes SEJ is the *conjunction* of *k* predicates (each of which corresponds to the condition for a 1-attribute SEJ), the order of these *k* predicates in the SQL statement will significantly affect the cost of the query [**4**]. To optimize the performance for *k*-attributes SEJ, we modify the order on these *k* predicates by referencing the size of each auxiliary table, and the join selectivity of each pair of join attributes, such that the total cost of accessing *k* auxiliary tables is minimum.

Finally, we evaluate the efficiency of the proposed approaches by performing intensive experiments on synthetic datasets with various characteristics.

The paper organization is given as follows. Section 2 provides a review of related work and how the SEJ is positioned in this context. Our proposed approach is shown in Section 3. The experiments and results are reported in Section 4. Finally, we provide concluding remarks in Section 5

2. Related Work and Discussion

The problem of joins with inexact matching of corresponding data, approximate string joins, has already been addressed by many authors experimenting with a variety of approaches. Most of them search for solutions that can be effectively executed by the existing and widely used traditional relational database environments, RDBMS and SQL, without any necessity for any special purpose programming.

The problem was addressed firstly in [**5**]. The main concept behind this approach is to decompose a string attribute value into substrings called q-gram [**6**] which has been used widely in string matching context. The length of substrings is specified by q value. For example, if q is set at 3, a string “John" can be decomposed into the set of substrings {“##J", “#Jo", “Joh", “ohn", “hn$", “n$$"}. When matching two strings, a distance function based on the edit distance is computed on the two sets of their substrings. A number of q-gram related propositions were proposed to reduce the cost of edit distance computation.

In [**7**], an approximate string join approach based on cosine similarity computation was proposed. In this work, string data of each attribute (which can be composed of many terms) is represented as a multi-dimensional vector. For each attribute, the number of dimensions is the number of all possible terms used in the attribute (a string may consist of multiple terms). For each dimension, a magnitude is assigned according to the significance of the corresponding term; an infrequent term will have a higher value, while a common term will have a lower value. For example, a string “Information Technology and Electrical Engineering" in an affiliation attribute with eight possible terms can be represented as an 8-dimensional vector (0, 0, 0.5, 0.2, 0.6, 0.4, 0, 0), where 0.5, 0.2, 0.6 and 0.4 represent the significance of the terms Information, Technology, Electrical and Engineering respectively. When strings are to be matched, the cosine similarity function is then computed. If the cosine similarity of two strings is higher than a pre-specified threshold, they can be matched. For example, if a specified threshold is set at 0.4, the above string and another string “Information Engineering" represented as a vector (0, 0, 0.5, 0, 0, 0.4, 0, 0) can be matched because their cosine similarity is 0.41 (0.5x0.5+0.4x0.4). Clearly, one can build many examples where such a matching will not deliver high quality results, mainly due to an evident lack of dependency measures between the terms.

Also, it is intrinsically difficult to allocate the best values for significance measures, and any allocation will be regarded as highly subjective. Also, in the same work, a sampling based computation has been used to reduce the computational cost of the function. Instead of computing a similarity function for every pair of the join candidates, which is an expensive operation, insignificant candidates will be sampled out, with only the promising pairs computed. In the same way as in [**5**], this work does not require any modification to RDBMS. Only auxiliary tables are required to be created and they are used to drive the matching mechanism.

In [**8**], web data integration was proposed as another application for approximate string joins. In that work, more extensive theoretical aspects and experimental results were presented. Furthermore, many combinations of distance functions were discussed, for example, edit distance, block edit distance, cosine similarity with words as terms, and cosine similarity with q-grams as terms. As a result, the last type of combination, cosine similarity with q-grams as terms, has been suggested as the best distance function because it can capture all types of mismatch, including spelling errors, insertion and deletion of short or common words and variations of word order.

An attempt to include semantic knowledge into the approximate string join computation (rather than purely structural string comparison as indicated above), is presented in [9]. To improve similarity-based joins, two terms which share the same semantic meaning, but have too much difference in their representations (e.g. “MCI" and “Worldcom" which refer to the same company), are formed as a pair in an auxiliary table by human experts. Subsequently, in the pre-processing phase of a join, each tuple is extended by associating its value with the pairs in the auxiliary table which have a “high" similarity score. For example, consider a tuple with a value “MCI Corp". After the pre-processing phase, the value will be extended to (“MCI Corp", “MCI", “Worldcom") by associating the value with a pair of (“MCI", “Worldcom"). When answering a matching query, the attribute values of the query string are also extended by using the auxiliary table in the same way as in the pre-processing phase. Suppose that an attribute value “Worldcom Corp" is given in the query, it will be extended to (“Worldcom Corp", “Worldcom", “MCI"). Consequently, it can be matched with the tuple above, with the “MCI Corp" value, by traditional cosine similarity computations. Although only briefly discussed in this paper, this approach can remedy the shortcomings of previously similarity-based approaches by applying semantic knowledge. Similar to our work, this work also addresses the synonym mismatch problem by applying semantic knowledge. However, unlike our work, which is to efficiently compute SEJ, the focus of that approach is on how to use semantic knowledge to pre-process two relations such that later approximate join approaches can achieve more accurate results without additional operations.

3. Semantically Equal Join Approach

3.1 Preliminaries

Let us first consider two relations $R_1(A_1,...A_m)$ and $R_2(B_1,...B_n)$ such that attributes A_1 and B_1 have the same domain *Dom*, where *Dom* is a set of strings.

Definition 1. ***Semantic equivalence****, denoted as $\cong$, is a relation defined on a given domain Dom such that for every a, b $\in$ Dom, a $\cong$ b iff a is a synonym of b. Also, we define that a and b are strict semantically equivalent, denoted as a $\sim$ b, if a$\cong$ b and a $\neq$ b.*

A_1
USA
United States
GBR
France

B_1
The States
United Kingdom
France

a) Join attributes

{(USA, USA), (United States, United States),
(The States, The States), (GBR, GBR), (United
Kingdom, United Kingdom), (France, France)}
$\cup$
{(USA, United States), (United States, USA),
(USA, The States), (The States, USA), (United
States, The States), (The States, United States),
(GBR, United Kingdom), (United Kingdom, GBR)}

b) Semantic equivalence relation

Figure 2: Semantic Equivalence

Example 1. Suppose that Figure 2a gives the values of join attributes A_1 and B_1 which are the results of projecting the relations R_1 and R_2 respectively. *Dom* in this example is the string set {USA, United States, The States, GBR, United Kingdom, France}. According to the standard convention used for expressing country names, we can naturally define the semantic equivalence relation $\cong$ on *Dom* as Figure 2b. Also observe that in this example, the strict semantic equivalence relationship is shown as the right-hand-side of the union operator.

Definition 2. Given a relation of semantic equivalence $\cong$, **semantically equal join (SEJ)** is a theta join operator in the relational algebra such that the theta comparison operator is $\cong$. Further, the semantically equal join operator is denoted as $\tilde{\bowtie}$

To perform the semantically equal join between two tables R_1 and R_2 on attributes A_1 and B_1, the semantic equivalence relation on domain *Dom* needs to be predefined. Since it is straightforward to observe that a string can semantically join with itself, we define a relation *Au*, called *auxiliary table,* to only specify the relationship of strict semantically equivalent, i.e., to give the information of synonyms in *Dom*.

According to Definition 1, it is easy to observe that semantic equivalence, $\cong$, is an *equivalence relation*[1]. Formally, given a relation of semantic equivalence relation $\cong$ on *Dom*, we can *partition Dom* into a set of *equivalence classes* C_i, *i*=1…*l* such that for any C_i, there does not exist elements $a, b \in C_i$ holding the condition of $a \ncong b$. Note that only the class C_i with $|C_i| > 1$ provides the information of synonyms. We define the set Ω of *non-trivial equivalence classes*, formally, $\Omega = \{C_i \mid |C_i| > 1\}$ where *i*=1…*l*. According to the above discussion, *Au* can be regarded as the representation of the set Ω.

Let us further consider the Example 1. Given the relation $\cong$, *Dom* is partitioned into three equivalence classes: $C_1 = \{$USA, United States, The States$\}$, $C_2 = \{$GBR, United Kingdom$\}$, and $C_3 = \{$France$\}$. The

[1]As an equivalence relation, "=" has the following property: 1) $a = a$ for all $a \in Dom$,
2) $a = b \rightarrow b = a$ for all $a, b \in Dom$
3) $(a = b) \wedge (b = c) \rightarrow a = c$ for all $a, b, c \in Dom$.

set Ω of non-trivial equivalence classes is $\{C_1, C_2\}$.

After introducing the key concepts, we generalize the problem of semantically equal join to *k* attributes and give the formal problem statement as follows.

Problem statement: Let $R_1(A_1,...A_m)$ and $R_2(B_1,...B_n)$ be two relations such that attributes $R_1.A_i$ and $R_2.B_i$ (*i*=1…*k*) have the same domain Dom_i, where Dom_i is a set of strings. For each Dom_i (*i*=1…*k*), let $\cong^i$ be the equivalence relation defined on Dom_i and Au_i be the auxiliary table that represents the corresponding strict semantically equivalent relationship. The semantically equal join (SEJ) problem is: given relations R_1, R_2 and auxiliary table $Au_1,..., Au_k$, express $R_1 \overset{\sim}{\bowtie}_{A_1 \cong^1 B_1,...A_k \cong^k B_k} R_2$ by a single SQL statement.

In the rest of Section 3, we will first study how to build an SQL statement to *efficiently* compute SEJ on one attribute, then discussion is extended to SEJ on *k*-attributes (*k*>1).

3.2 Schema of Auxiliary Table and Semantically Equal Join

Firstly, we propose a structure of auxiliary tables, called the group-based approach for the schema of the auxiliary table. In this approach, given the set Ω, the auxiliary table *Au* is defined as $Au(Gid, P) = \{(i, a) \mid C_i \in \Omega$ and $a \in C_i\}$. In Example 1, the auxiliary table is given in Figure 3. With this schema, we can build the primary index on the attribute *P* for efficient access because there can not exist a string that belongs to two different synonym groups.

GID	P
1	USA
1	The States
1	United States
2	GBR
2	United Kingdom

Figure 3: The example of group-based auxiliary table

Given the schema of *Au*, the RA expression for the semantically equal join for this schema is:

$$R_1 \overset{\sim}{\bowtie} R_2 \equiv (R_1 \bowtie_{A_1=B_1} R_2) \cup$$

$$((R_1 \bowtie_{A_1=P} Au) \bowtie_{Au.GID=Au'.GID} (R_2 \bowtie_{B_1=P} Au')) \quad (1)$$

where Au' is the copy of Au.

3.3 Query for Implementing SEJ

Given the relational algebra operation in Equation (1), there are multiple ways to build an SQL statement to compute the semantically equal join. Although generally, different queries may have the same execution plan in a given DBMS, in many cases, the way to write a query will affect the execution plan and the query performance. In this section, we propose an SQL statement which can efficiently compute semantically equal join. Rather than considering query optimization techniques which are dependant on the RDBMS, we only put our efficiency concern at the SQL level.

3.3.1 SQL Statements

According to Equation (1), we can directly translate the RA into an SQL statement, which is shown in Figure 4a. Since part of this SQL statement requires three join operations to link four tables, we call it *three-join-based SQL statement*.

```
(SELECT * FROM R1, R2 WHERE A1=B1)
UNION
(SELECT * FROM R1, R2, Au Au1, Au Au2
WHERE (A1=Au1.P AND B1=Au2.P AND
Au1.id = Au2.id))
```

a) Three-join-based

```
SELECT * FROM R1, R2 WHERE A1=B1
OR ((SELECT Gid FROM Au WHERE
P=A1) = (SELECT Gid FROM Au WHERE
P=B1))
```

b) Sub-query-based

Figure 4: SQL statements

Intuitively, the cost of joining four tables is expensive. Therefore, to improve the efficiency, we propose another SQL statement in Figure 4b, called *sub-query-based SQL statement*. The semantic meaning behind this query is that two tuples can be joined together if either of the values of the join attributes are the same, or they belong to the same group in the auxiliary table. Note that when both values of A_1 and B_1 are not in the auxiliary table, the condition after 'OR' will compare two Null values. In the standard SQL, the result of Null=Null is 'unknown', which will not make two tuples join.

3.3.2 Cost Analysis

We briefly discuss the I/O cost for computing the SEJ in this section.

Let us first consider the three-join-based SQL. In Figure 4a, we can see that the query consists of three operations: equal join, operation of three-join, and union. It is not hard to observe that this query will generate a significant amount of inter-mediate results from joins and require the external sort to remove duplicates for the union operation, both of which will lead to high I/O cost.

For the sub-query-based approach, the query plan is much simpler: When R_1 and R_2 are joined, for any pair of tuples which can not equally join, the auxiliary table is accessed by the primary index (on the attribute *P*) twice to check the semantic join condition. Note that in practice, an auxiliary table is not very large and often can be cached in the buffer. In this case, the I/O cost of sub-query-based SQL is close to the cost of an equal join of R_1 and R_2.

For the above discussion, we can see the sub-query based approach requires less disk I/O when the size of *Au* is small. This will be also demonstrated in the experiment at section.

3.4 k-Attributes Semantically Equal Join

Now, we consider the task of semantically joining two relations on *k* attributes, as is described in the **problem statement** in Section 3.1.

As the join condition of *k*-attributes SEJ is the conjunction of the conditions for the single attribute joins, we can readily extend the previous SQL statement shown in Figure 4 into the statement in Figure 5.

Since the join conditions are linked by *AND*, if any set of condition is not satisfied, it is not necessary to check the other join conditions of other attributes. Apparently, we can order the join conditions sets to minimize the cost of accessing auxiliary tables. Intuitively, two factors need to be considered: 1) the size of auxiliary table Au_i 2) the join/SEJ selectivity of each pair of join attributes.

```
 SELECT * FROM R1, R2
WHERE (A1=B1 OR
((SELECT Gid FROM Au1 WHERE
P=A1)=(SELECT Gid FROM Au1 WHERE
P=B1)))
…
```

AND(A_k=B_k OR
((SELECT Gid FROM Au_k WHERE P=A_k)=(SELECT Gid FROM Au_k WHERE P=B_k)))

Figure 5: SQL for *k*-semantically equal join

It is well-known to the database community that the query performance can be adjusted by ordering join conditions linked by *AND*. However, driven by the specialty of a new type query, we discuss this ordering problem on the context of SEJ with the purpose of minimizing the cost of auxiliary table access.

Let p_i be the number of pages of the auxiliary table Au_i where i=1,...,k. Given relation R_1 and R_2, let $\hat{s}_{i_1,\ldots,i_l}$ (i_j is the index of the attribute $1 \le i_j \le k$) be the *semantically equal join selectivity* of R_1 and R_2 on *l* attributes. Formally, $\hat{s}_{i_1,\ldots,i_l}$ is equal to: $\frac{\left|R_1 \tilde{\bowtie}_{A_{i_1} \cong B_{1_1},\ldots,A_{i_l} \cong B_{1_l}} R_2\right|}{\left|R_1 \times R_2\right|}$. Similarly, we define the *equal join selectivity* $s_{i_1,\ldots,i_l}$ as: $\frac{\left|R_1 \bowtie_{A_{i_1} = B_{1_1},\ldots,A_{i_l} = B_{1_l}} R_2\right|}{\left|R_1 \times R_2\right|}$.

Given an order of join attributes $i_1,\ldots,i_k$, under the assumption that the join attributes are independent, the total I/O cost on access *k* auxiliary tables is:

$$|R_1||R_2| * [(1-s_{i_1})p_{i_1} + \hat{s}_{i_1}(1-s_{i_2})p_{i_2} + \ldots + \hat{s}_{i_1,i_2,\ldots i_{k-1}}(1-s_{i_k})p_{i_k}]$$

Also due to the assumption that the join attributes are independent, we have $\hat{s}_{i_1,\ldots i_l} = \prod_{j=1\ldots l} \hat{s}_{i_j}$. So the ordering task can be presented as:

Given the sizes of auxiliary tables $p_1, p_2, \cdots, p_k$, the equal join selectivity of each pair attributes $s_1, s_2, \cdots s_k$, and the semantically equal join selectivity of each pair attributes $s_1, s_2, \cdots s_k$, are ordered to find an order $i_1, i_2, \cdots, i_k$ such that the **cost function**

$$[(1-s_{i_1})p_{i_1} + \hat{s}_{i_1}(1-s_{i_2})p_{i_2} + \ldots + \hat{s}_{i_1,i_2,\ldots i_{k-1}}(1-s_{i_k})p_{i_k}]$$ is minimized.

To find the order with minimum cost, theoretically, we need to compute the costs for *k*! orders. However, based on the cost function, we can apply some heuristics to speed up the search.

The equal join selectivity or semantically equal join selectivity is often small. Considering the cost function, assume that the values of $(1-s_i)p_i$ for $i=i_1,\cdots,i_k$ are not significantly different, we can expect that the cost of accessing *i*-th auxiliary table is less than that of accessing the $(i+1)$-th auxiliary table in a manner of magnitude. Intuitively, to minimize the total cost, it could be helpful to first select index *i* with minimum $(1-s_i)p_i$. Based on this idea, we can ignore the semantically equal join selectivity and order the index accordingly according to the value $(1-s_i)p_i$.

Once the optimum order *O* of index on the join attributes is found, the query in Figure 5 can be optimized by re-ordering the join conditions based on *O*.

4. Experimental Results

In this section, to evaluate the efficiency of SEJ, we report the result of the performance study on synthetic datasets.

4.1 Experimental Configuration

System resource: The experiments are conducted on an 800 MHz Intel Pentium III with 256 megabytes main memory, running a commercial RDBMS (Oracle 8i). All invented methods are implemented using standard SQL.

Datasets: In our experiments, all the relations (R_1,R_2, and auxiliary table Au) are generated by different random variables, each of which follows the *uniform distribution*. For brevity, we only show how to generate relations for 1-attribute SEJ. The discussion can be readily extended to k-attributes SEJ.

There are two controlled parameters for the auxiliary table, the cardinality $|Au|$ and the number g of semantically equivalent groups. Given $|Au|$ and g, we generate the group-based auxiliary table $Au(Gid,P)$ by randomly assigning $|Au|$ distinct strings into g groups. Based on the uniform distribution, $\frac{|Au|}{g}$ gives the expected number of string values in each group, which is denoted as v.

For the two relations R_1 and R_2, their schema are designed as $R_1(A_1, A_2)$ and $R_2(B_1, B_2)$ with A_1 and B_1 to be the join attributes. Further, we consider the instances in each relation. Again for simplicity, we set the constraints that the values of attribute A_1 are exactly the same as the values of attributes B_1 Considering the attribute A_1 (B_1), its values consist of three parts: the S part which has the attribute values in the auxiliary table (i.e., can semantically match the values in the other relation), E part which has the values that can only exact match the values in the other relation, and the N part, which has the values that cannot match any values in the other relation. Let $S\%,E\%,N\%$ ($S\%+E\%+N\%=1$) be the percentages of the values in part S, part E, and part N respectively, and ns,ne,nn be the numbers of distinct values in these there parts. Given the above parameters, the join relations are generated by following the equal distribution. For example, in S part, we can expect each distinct value appears $\frac{|R|*S\%}{ns}$ times, where $|R|$ is the cardinality of R_1 (R_2).

4.2 Experimental Results and Discussion

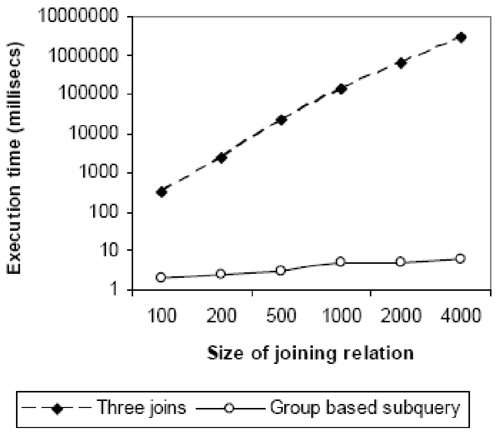
1-Attribute SEJ Results and Discussion

Effect of join relation size |R|: In this experiment, we increase $|R|$ from 100 records to 4,000 records to examine the performance of two types of query with different sizes of join relation. The other parameters of the join relations are set as: $S\%=80\%$, $E\%=0$, $N\%=20\%$, $ns=10$, $ne=0$, $nn=3$. For auxiliary table, we set $|Au|=10$ and $g=3$.

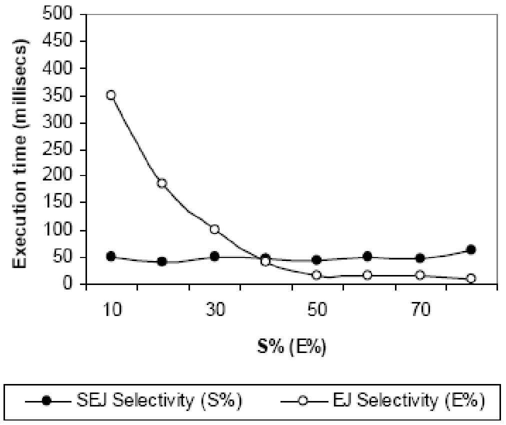
The performance of 1-attribute SEJ by different types of queries is presented in Figure 6a. It can be seen that the three-join-based approach is very inefficient. Therefore, it cannot be used for a large amount of data. As discussed in Section 3, the SQL statement for three-join-based approach requires that four tables are joined, which leads to expensive I/O cost. Therefore, for the rest of this section, we will omit the result of the three-join-based approach and only discuss the group-based sub-query.

Effect of selectivity $\hat{s}$: The second consideration for 1-attribute SEJ is about its performance regarding the *semantic join selectivity* $\hat{s}$ of R_1 and R_2 where $\hat{s}$ is composed of two parts, *strict semantic join selectivity* s' (i.e. the selectivity for the pairs of tuples that are matched based on strict semantically equivalent relationship) and *equal join selectivity* s (introduced in Section 3.4). By definition, we always have $\hat{s} = s + s'$. Note here that we slightly change the way of generating synthetic datasets in this part. The records in S part are allowed to only be joined through an auxiliary table (i.e., no equal join in this part). In this way, due to the equal distribution, the strict semantic join selectivity is $\frac{(S\%)^2}{g}$, where g is the number of groups in *Au*. Similarly, given *E*% and *ne*, the equal join selectivity can be estimated as $\frac{(E\%)^2}{ne}$. In the experiments, the strict semantic join selectivity *s'* and equal join selectivity can be adjusted by changing parameters *S%*, *ns*, *E%*, and *ne*. Both experiments use some common parameters which are the size of join relations |*R*| at 10,000 tuples with *ne* at 5 and *nn* at 100. Also, for both experiments, we set the size of auxiliary table |*Au*| at 100 with *g* at 25, which means that *v* is set at 4.

To evaluate the impact of s', we fix *s* by always setting *E*%=10% and *ne*=5. We increase the *strict semantic join selectivity* by increasing *S*% from 10% to 80%. Note that when *S*% is increased, we also decrease *N*% to keep the size of join relation the same. For evaluating the performance of SEJ regarding to equal join selectivity *s*, we fix the s' by always setting *S*%=10% and *ns*=100 and increase *E*% from 10% to 80%.



a) Effect of join relation



b) Effect of selectivity

Figure 6: Performance of SEJ

We firstly measure the performance of SEJ regarding to s', which is shown in Figure 6b. Obviously, the performance of SEJ is irrelevant to *strict semantic join selectivity*. This verifies our thoughts that the amount of time for auxiliary table access only depends on the equal join selectivity *s*. The reason is that for any two records $t_1 \in R_1$ and $t_2 \in R_2$ the auxiliary table needs to be accessed unless t_1 and t_2 can be equal joined. Also, in Figure 6b, we report the performance of SEJ when the selectivity *s* is increased. We can see that due to the same reason, the execution time decreases when *E*% is increased.

***k*-Attributes SEJ Results and Discussion**

In this experiment, we discuss the SEJ join in terms of the order of *k* attributes in an SQL statement. Particularly, we verify our cost analysis in Section 3.4 and demonstrate how the proposed heuristics help to quickly find an optimal order. As discussed previously, the following two factors are considered for the ordering: 1) size of auxiliary table $|Au_i|$ 2) join/SEJ selectivity. We generate join relations with three join attributes (*k=3*) of size 100,000 records. The characteristics and the estimated selectivities of the relations are shown in Table 1. As shown in Table 1, the first and the second attributes have the same |*Au*|, but they are different in selectivities. On the other hand, even if the third attribute has the lowest selectivity, it has the largest |*Au*|. The performance of all possible orders of join conditions for the dataset is evaluated.

The performance on *k*-attributes SEJ with all possible join condition orderings is reported in Figure 7. On the x-axis, we put the ascending order for all theoretical I/O costs (for different ordering) computed from the **cost function**. The result shows that the order of actual cost is accordant with the order of estimated cost, theoretically. This demonstrates the correctness of our previous analysis. Compared with the worst case order **3-2-1**, the best order **1-2-3** has better performance, by 17.9%. As mentioned in Section 3.4, we can also compute the best order by computing estimated cost $(1-s_i)p_i$, in which the result is also the order **1-2-3**. Another interesting observation is that the order **2-1-3** has more cost than the order **1-3-2**, which shows that the SEJ selectivity should also be considered, not only the size of auxiliary tables. On the other hand, ordering only by SEJ selectivity can not perform very well, as presented by the order **3-1-2**.

Table 1: Characteristics and Selectivities of Experimental Dataset

Attribute	Auxiliary table			Join relation						Selectivity		
	|*Au*|	*g*	*v*	*S*%	*E*%	*N*%	*ns*	*ne*	*nn*	s'	s	$\hat{s}$
1	200	50	4	10%	60%	30%	200	10	100	0.2%	3.6%	3.8%
2	200	50	4	50%	10%	40%	200	10	100	5.0%	0.1%	5.1%
3	400	50	8	40%	20%	40%	400	10	100	3.2%	0.4%	3.6%

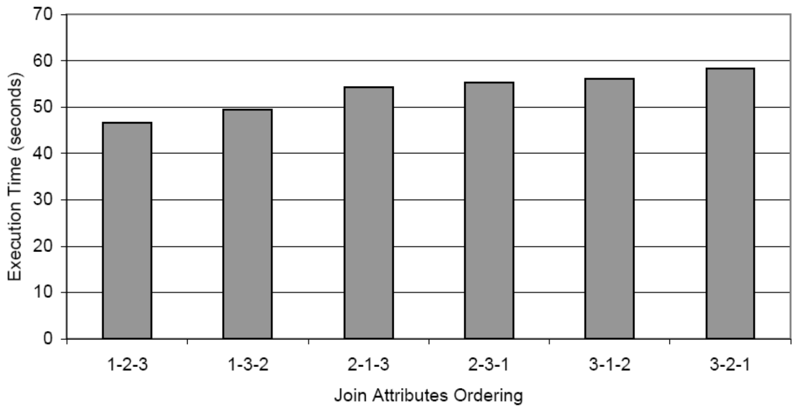


Figure 7: Performance of k-Attribute SEJ.

5. Conclusion

In this paper, we have addressed the problem of synonym mismatch in the context of data integration, where the data from different sources are semantically equivalent, but the data representation is different. We have introduced a new relational algebraic operation, called semantically equal join (SEJ), which can be generally applied for integrating two relations based on predefined equivalence relation(s). The originality and contributions of our work include the following aspects:

1) we have introduced the semantic equivalent relationship to resolve the synonym mismatch problem. In fact, this study can also complement previous work of approximate join that mainly focuses on resolving typo mismatch. 2) We have formally defined the concept of SEJ and proposed an efficient approach to implement the SEJ operator in the standard RDBMS by a single SQL statement. 3) By extensive experiments we have demonstrated performance of our approach for SEJ.

6. References

[1] Halevy, A.Y., Ashish, N., Bitton, D., Carey, M., Draper, D., Pollock, J., Rosenthal, A., Sikka, V., Enterprise Information Integration: Successes, Challenges and Controversies, In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM Press, pp.778–787, 2005.

[2] Rahm, E., Bernstein, P.A., A Survey of Approaches to Automatic Schema Matching, the VLDB Journal,Vol. 10, pp.334–350, 2001.

[3] Colomb, R.M., Orlowska, M.E.: Interoperability in Information Systems. Information Systems, Journal,Vol.5, pp.37–50,1994.

[4] Orlowski, M.W., On Optimisation of Joins in Distributed Database System, In: Future Databases, pp.106–114,1992.

[5] Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D., Approximate String Joins in a Database (Almost) for Free, In Proceedings of 27th International Conference on Very Large Data Bases, Morgan Kaufmann, pp.491–500, 2001.

[6] Ullmann, J.R., A Binary *n*-gram Technique for Automatic Correction of Substitution, Deletion, Insertion and Reversal Errors in Words. The Computer Journal, Vol. 20, pp.140–147, 1976.

[7] Gravano, L., Ipeirotis, P.G., Koudas, N., Srivastava, D., Text Joins for Data Cleansing and Integration in an Rdbms, In Proceedings of the 19th International Conference on Data Engineering, IEEE Computer Society, pp.729–731, 2003.

[8] Gravano, L., Ipeirotis, P.G., Koudas, N., Srivastava, D., Text Joins in an Rdbms for Web Data Integration, In Proceedings of the Twelfth International World Wide Web Conference, pp.90–101, 2003.

[9] Koudas, N., Marathe, A., Srivastava, D., Flexible String Matching Against Large Databases in Practice, In Proceedings of the Thirtieth International Conference on Very Large Data Bases, Morgan Kaufmann, pp.1078–1086, 2004.