Implementation and comparison of DUMAS data integration algorithm in PostgreSQL management system algorithms

Final report of project in
CS5513 Advanced Database Management 2019 Spring

Xiang Yu #yuxiang@ou.edu; Linsheng He #helinsheng@ou.edu

Implementation and comparison of DUMAS data integration algorithm in PostgreSQL management system algorithms

Xiang Yu
School of Computer Science
University of Oklahoma
Norman, OK, US
yuxiang@ou.edu

Linsheng He
School of Electrical and Computer Engineering
University of Oklahoma
Norman, OK, US
helinsheng@ou.edu

ABSTRACT

With the exponential growth of databases in the Internet age, data integration has great potential for development in the information era.

This project aimed to research and analyze these techniques of data integration, the context of which was the development of an application for merging relational databases. we implemented a data integration algorithm based on PostgreSQL database according to the description of Duplicate-based Schema Matching (DUMAS). And on this basis, in the same databases, compared the time consumption with a data integration algorithm born in recent years, which is label-based schema matching algorithm. Finally, under different threshold choices, discuss the differences between the two algorithms and give suggestions for improvement.

KEYWORDS

Data Integration, PostgreSQL, DUMAS

1 Introduction

Since the birth of the computer, the amount of usage has grown rapidly, and today it is already an integral part of society. As the use of media such as the Internet increases and storage prices drop, the amount of data we process is growing. The database is no exception. Since the beginning of the 1970s, the use and amount of data stored in relational databases have followed this trend. With the widespread use of relational databases, one of the toughest problems is the integration of data.

Nowadays, more and more devices are connected to the internet, and data is a very important resource for business companies. Better data analysis can help companies to make a better marketing strategy. However, in order to do data analysis, the companies have to collect data from multiple sources. For example, Amazon needs to collect data from the website, user-application from the phone call, so Amazon could show the right product advertisement for a potential customer. Collecting TV advertisement event data from YouTube to check which category's products fit for a specific customer. But those data from a different source so that they have different data format and unit. Therefore, we need data integration to transform those data sources into the same data format which is easier and better for the company to do data analysis. Data integration is defined as combing different data set from different data sources and provide a new single data format to end-user [1]. Because the database is flexible and heterogeneous, therefore, data integration is very important for business completive in commerce field. Therefore, accuracy, time-consuming and space complexity are very important for data integration. They need to minimize time and hardware resources to integrate heterogeneous data sources and get more accuracy data format to make better marketing strategy so that they can stand in strong business competition stably [2].

1.1 Data integration

Data integration is an abstract topic about combining data from different data sources, in order to improve our understanding of data integration and find out the approaches to implement data integration. To summarize, the goal of a data integration system is to offer uniform access to a set of autonomous and heterogeneous data sources [3]. There are several challenges in the data integration field showing below:

1. Numbers of query: The focus of most data integration systems is on querying disparate data. sources. However, updating the sources is certainly of interest.

2. Numbers of sources: Data integration is already a challenge for a small number of sources (fewer than 10 and often even 2), but the challenges are exacerbated when the number of sources grows.

3. Heterogeneity: A typical data integration scenario involves data sources that were developed independently of each other. As a consequence, the data sources run on different systems: some of them are databases, but others may be content management systems or simply files residing in a directory. The sources will have different schemata and references to objects, even when they model the same domains. Some sources may be completely structured (e.g., relational databases), while others may be unstructured or semi-structured (e.g., XML, text).

4. Autonomy: The sources do not necessarily belong to a single and even when they do, they may be run by different sub-organizations. Hence, we cannot assume that we have full access to the data in a source or that we can access the data whenever we want, and considerable care needs to be given to respecting the privacy of the data when appropriate. Furthermore, the sources can change their data formats and access patterns at any time, without having to notify any central administrative entity.

1.2 Objective of this study

In this study, we propose to use two data integration algorithms to compare the pros and cons of these algorithms by using time-consuming evaluation metrics. In other to implement the data integration algorithm inside the DBMS, we are going to use PostgreSQL which is an open source.

PostgreSQL is open source and modifiable. Through doing research for comparing two existing different algorithms for data integration from multiple sources in PostgreSQL. Data integration is consisting of the following processing: data extraction, data transformation, data loading to the target system. Our research is that using two different algorithms to test time complexity and data quality. Data extraction is extracting data from a source such as a web and put it into the data warehouse, Data transformation converts the associated interdisciplinary data entities from a source format to a target format. Finally, we need to combine data collected from different sources.

2 Related work

There is some various architecture of data integration system, a lot of people build their data integration system falls on the spectrum between virtual integration and warehousing [4]. There was a wide mix of styles and methods used for integration, for instance: Clio, Merge and BDK. Rather than utilizing the data in the tables, Clio tackles the problem by processing concepts in the schema. Merge is an algorithm which integrates two models based on their common correspondences. BDK's pre-integration consists of subsuming the database schema into a weak entity ER model. Among the algorithms detailed above, Clio, Merge, and BDK takes a semantic approach from a more logical and mathematical perspective. Central to these three are the notions of real life "concepts" or "objects", that is, they are trying to perform the integration in a similar way to which a human may approach it; first identifying which concepts relate to one another in the two schemas, and then finding a way to combine these into one target.

The logic-based algorithms are not just available for use on relational databases but also any data source that can be transposed into an appropriate model. As the algorithms solely deal with this model and no actual data, a wide range of sources are possible. Conversely, it could be said that not using the instance data in the integration is an opportunity wasted. The model-based approaches require some form of user interaction to confirm what they assume are the same object, is in fact correct.

Besides, the data-based approach has proved very effective for schema alignment in databases based on a mere handful of duplicate tuples out of a very large dataset.

3 Proposed work

In order to approach the data integration, in this study, we are going to use the reference of DUMAS [5] which is a data-based approach to do the data integration. The DUMAS algorithm is a mediator data integration which use mediator schema to access multiple data source and get the combined answer back instead of copy multi data and combine them into a new database.

3.1 Duplicate-based Schema Matching (DUMAS)

Most of the data integration application requires the matching the schema between data set, and the DUMAS approach performs horizontal match: compare tuple from a data to other data set, to find out the pair similar tuples, which is called detecting duplicates. After detecting there are duplicates, the system will do a matching schema approach, because the same or similar values of duplicates imply the corresponding attribute schema. However, detecting duplicates are difficult, the pair tuple may not have subset attribute in common, however, their attribute values are the same, which maybe mislead the system in detecting duplicates. The author Mikhail and Raymond mention that the textual similarity in edit distant could improve duplicates detection [6].

The second main part of the DUMAS is schema matching. After identifying the duplicates, the schema matching will find alignment and attribute correspondences between two schemas. There are some different schema matching approaches, such as schema-based method which is using labels, types and other schema information to find a similar attribute, instance-based method combines attribute label to derive data matching. And the third popular method to approach schema matching is Internet Learning Agent.

$\boldsymbol{R}$	$FirstName$	$LastName$	Sex	$Phone$	Fax
r_1	John	Doe	m	(408) 7573339	(408) 7573338
r_2	Joe	Smith	m	(249) 3615616	(249) 2342366
r_3	Suzy	Klein	f	(358) 2436321	(358) 2436321
r_4	Sam	Adams	m	(541) 8127100	(541) 8121164
r_5	Mark	Spitz	m	(901) 8319311	(901) 8612382
r_6	Jim	Beam	$\perp$	(782) 1238957	(781) 1883744
r_7	Kate	Moss	f	(124) 9654565	$\perp$
r_8	Sam	Wong	f	(124) 4955670	(999) 9999999
r_9	John	Dean	m	(369) 3663624	(367) 3663625

$\boldsymbol{S}$	LN	Acc	Tel	OS
s_1	Douglas	jdouglas	(408) 9182043	XP
s_2	Dean	jd	(369) 3663624	XP
s_3	Klein	littlesue	(358) 2436321	UNIX
s_4	Adams	sam	(541) 8127100	W2000
s_5	Wong	kate	(923) 6363443	Linux
s_6	Kurz	itsme	$\perp$	UNIX

Figure 1. Relations R and S with intentional and extensional overlap

To illustrate the general idea, consider the example depicted in Figure 1. Both relations R and S contain information about people but are differently structured: Relation R has attributes for the first name, last name, gender, phone number, and fax number, while relation S contains attributes representing last name, user account, phone number, and operating system. The goal of schema matching is to detect correspondences between semantically

related attributes. In the example scenario only, the following correspondences exist:

LastName == LN

Phone ==Tel

The problems described above can be solved by following the DUMAS approach, which exploits duplicate tuples for schema matching. Different representations of the same real-world object are called duplicates. In the example in Figure 1, tuples r3, r4, and r9 in R represent the same entities as tuples s3, s4, and s2 in S, respectively. Those duplicates provide valuable information that can be used for schema matching. Two duplicates, namely (r4, s4) and (r9, s2), also provide hints that help in distinguishing Phone and Fax: Both r4 and r9 have different values for Phone and Fax, and the Phone values equal the Tel values in the respective matching tuple. In this study, we show how to extract attribute correspondences from duplicate tuple pairs.

In principle, any duplicate-based schema matching must proceed in two steps:

1. Duplicate detection: A few duplicate tuples are found in the databases, and

2. Matching: Attribute correspondences are extracted from the duplicates.

Algorithm 2: DUMAS Schema Matching Algorithm

Input: Source schema s; Target schema t
Output: Schema Matching between s and t

```
1  match ← initialMatch(s,t);
2  sourceTree ← new DerivationTree(s,match);
3  targetTree ← new DerivationTree(t,match);
4  repeat
5      nodeMatches ← ∅;
6      lastMatch ← match;
7      extend(sourceTree); extend(targetTree);
8      if numPending(sourceTree) + numPending(targetTree) == 0 then
       Break;
9      foreach pend in getPending(sourceTree) ∪ getPending(targetTree)
       do
10         parent ← parent(pend); pPartner ← partner(parent);
11         ppMatch ← match(table(pend), table(pPartner));
12         if sanityCheck(ppMatch) then
13             nodeMatches ← nodeMatches ∪ ppMatch;
14         end
15         nodeMatches ←
           nodeMatches ∪ childMatches(pend, pPartner, ppMatch);
16     end
17     nodeAssignment ← maximumWeightMatching(nodeMatches);
18     foreach match in nodeAssignment do
19         sNode ← getSource(match); tNode ← getTarget(match);
20         matching(sNode) = match; matching(tNode) = match;
21         if isLeaf(sNode) then activate(sNode);
22         if isLeaf(tNode) then activate(tNode);
23     end
24     match = globalMatching(sourceTree);
25 until |lastMatch| ≥ |match|;
26 return lastMatch;
```

Figure 2. Pseudo-code of DUMAS algorithm

The DUMAS algorithm to compute a matching between complex schemata is summarized in Figure 2. In the beginning, the initial matching is constructed the derivation trees for the source and target schemata are constructed, this is a duplicate method.

Lines 4 through 25 describe the iterative part. It starts by initializing two variables: nodeMatches is a set of matchings between nodes, which are constructed in the current iteration. At the beginning of the iteration, it is empty. The variable lastMatch contains the schema matching that has been constructed in the previous iteration. In the first iteration, the initial matching is used.

Afterward, active nodes of the source tree and the target tree are extended. A node in a derivation tree is an active node if it has produced new correspondences in the previous iteration. Active nodes can easily be extracted from a derivation tree: Active nodes are leaf nodes which have a partner node and a node matching. A node is extended in two steps: First, new join tables are created by joining the node's table with the neighboring base table. Neighboring tables are base tables that are not part of the node's table but are connected to one of its constituent base tables by a foreign key dependency. For each of the neighboring tables, a new join table is created that is the result of joining the node's table with the neighboring table2. Second, a child node that is the child node of the active node is created for each new join table. The newly created nodes are called pending nodes. The iteration stops if the number of pending nodes is zero. This could occur in small schemata when the active nodes represent the whole schema, and thus, no base tables can be joined.

3.2 Label-based Schema Matching

The label-based schema data integration method within the WInte.r [6] framework which supports end-to-end data integration processes by providing algorithms and building blocks for web data pre-processing, schema matching, identity resolution, as well as data fusion.

The WInte.r framework is fully usable out-of-the-box with implementations for all mentioned data integration tasks and provides highly customizable functionality that allows for the composition of sophisticated integration architectures. This allows researchers to focus on the actual task instead of starting from scratch and repeatedly implementing the same functionality in different projects. The WInte.r framework is written in Java and is available as open source under the Apache 2.0 license.

4 Methodology

4.1 Languages and dataset

In this study, Java was used to implement for Eclipse and coding. Java is a widely used language and is supported natively on the required platforms. Development-wise, Java has a number of fully-featured mature IDEs which are cross-platform.

PostgreSQL is an open-source relational database management system (RDBMS) emphasizing extensibility and standards compliance. It is also a powerful database system with a strong reputation for reliability, data integrity, and correctness [7].

Region	Country	Item Type	Order Priority	Order ID	Unit Cost	Total Revenue
Central America and the Caribbean	Antigua and Barbuda	Office Supplies	M	3.63E+08	524.96	4035548.37
Middle East and North Africa	Jordan	Fruits	C	5.22E+08	6.92	68342.25
Sub-Saharan Africa	Mali	Meat	L	2.86E+08	364.69	2887415.16
Middle East and North Africa	Somalia	Cereal	C	2.15E+08	117.11	142755.8
Middle East and North Africa	Kuwait	Snacks	L	4.9E+08	97.44	1045173

Figure 3. Examples of database I

Region	Sales Channel	Order ID	Units Sold	Total Cost	Total Profit
Middle East and North Africa	Offline	6.87E+08	8446	2224085	1468506.02
North America	Online	1.86E+08	3018	274426.7	190526.34
Middle East and North Africa	Offline	2.46E+08	1517	241840.1	145419.62
Asia	Offline	1.61E+08	3322	389039.4	294295.98
Sub-Saharan Africa	Offline	6.46E+08	9845	68127.4	23726.45

Figure 4. Examples of database II

The two databases trying to be integrated were collected from the Internet and both of them have a relationship with the information of the global product sales. The first database contains 502 data includes Region, Country, Item Type, Order Priority, Order ID, Unit Cost, and Total Revenue. And the second database contains 504 data includes Region, Sales Channel, Order ID, Units Sold, Total Cost, and Total Profit. A demonstration of two databases which contains the first 5 rows of the datasets and the titles are shown in Figure 3 and Figure 4.

4.2 TFIDF

In information retrieval, TFIDF, short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus [8]. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. TFIDF can be successfully used for stop-words filtering in various subject fields, including text summarization and classification.

The TFIDF measure has features that make it a reasonable choice for a tuple similarity measure:

1. Order independence: The TFIDF measure is a token-based measure, and thus, order-independent because each string is represented as a bag of tokens. Considered each tuple as a single string and applied an order-independent string similarity measure.

2. TFIDF weighting: The inverse document frequency, which is part of the TFIDF weighting scheme, gives a large weight to terms which appear infrequently. Thus, if a certain value appears in many tokens, it gets a relatively low weight, and thus, has a smaller effect on the similarity of two tuples.

3. Efficient top-k search: Efficient algorithms for detecting the k most similar strings exist and can be applied for duplicate detection.

The well-known TFIDF weighting scheme calculates the weight as a function of the term frequency. The weight $\omega'(s,t)$ of a term t in a string s as:

$$\omega'(s,t) = \log\left(tf_{s,t} + 1\right) \cdot \log\left(\frac{N}{df_t} + 1\right)$$

where $tf_{s,t}$ is the term frequency of t in s, N is the overall number of tuples, and df_t is the number of tuples in which t appears. The tuple similarity of two tuples r and s is calculated as:

$$tupsim(r,s) = \sum_{t \in r \cap s} \omega'(r,t) \cdot \omega(s,t)$$

4.3 Duplicate detection

Duplicate detection is inherently a problem with quadratic complexity: Each tuple in the first table must be compared with each tuple in the second table. Such an exhaustive search is clearly infeasible for larger data sets. Reducing the number of tuple comparisons is very important to make duplicate detection scalable. Our duplicate detection algorithm achieves this goal by detecting the K most similar tuple pairs.

Our algorithm considers each tuple as a single string and employs a string comparison metric to compare two tuple (strings). After comparing two tuples, one must determine if the two tuples are duplicates.

4.3.1 String comparison methods

Token-based similarity measures [9] interpret a string as a bag of words. The vector space model is widely used in the information retrieval community. In this model, each string is represented as a vector containing weights for each term of a (global) vocabulary. The details of steps are shown below:

Step 1. The first step of the algorithm is to take each row of the first table, treating the whole row as a string. We receive a score from the comparator instance and compare it to the threshold. We then add it to a "scoreboard" of size K. Once this process has been repeated, we are left with a list of the top K duplicates.

Step 2. From the top K duplicate tuples, we compare each field in the first table with each in the second. This results in a field similarity matrix; a set of SoftTFIDF scores for each field comparison. For these two tuples, this implies that any value above the token threshold corresponds to a match in those two columns. Varying the token threshold affects the results at this point; a lower threshold may incorrectly identify two columns s duplicates and conversely, too high may miss correct matches.

Using a TFIDF-based measure has several advantages. First, it is order-independent, which is important with respect to Problem 1. Second, by using the inverse document frequency, terms that occur in only a few tuples receive a higher weight.

A Duplicate detection demo from two datasets are shown in Table 1.

	Asia	3322	389039.4
Asia	1	0.05	0.01
C	0.2	0.08	0.03
142755.8	0.2	0.44	0.79

Table 1. Duplicate detection demo

4.4 Matching

In this paper, we only consider the simple (1:1) matches case in the schema matching step. The goal of this step is to deduce

attribute matches from K high-confidence duplicate pairs, which have been discovered in the duplicate detection step.

For comparing tuple fields, we use the Soft-TFIDF measure [9], a variation of the previously described TFIDF measure that also considers similar terms

$$fieldsim(a_i, b_j) = \sum_{t_a \in CLOSE(\theta, a_i, b_j)} \omega(a_i, t_a) \cdot \omega(b_j, t_b) \cdot termsim(t_a, t_b)$$

where:

$\text{CLOSE}(\theta, a_i, b_j) = \{t_a \in a_i | \exists t_b \in b_j, termsim(t_a, t_b) > \theta\}$

and,

$$termsim(t_a, t_b) = 1 - \frac{ed(t_a, t_b)}{\max(\{|t_a|, |t_b|\})}$$

where: (t_a, t_b) is the normalized edit distance and $\max(\{|t_a|, |t_b|\})$ being the longer of the two terms. $\text{CLOSE}(\theta, a_i, b_j)$ is the set of all terms in a_i that are > θ similar to a term in b_j.

We generate such a matrix Mk for each of the K duplicates and combine all matrices to produce the overall average similarity matrix M as input to the next step:

$$\text{M} = \frac{1}{K} \sum_{k=1}^{K} M_k$$

In summary, M stores average similarity scores that are accumulated over the field-similarities of the K most similar duplicates.

A Matching demo from two datasets are shown in Table 2.

	A	B	C
A’	0.92	0.07	0
B’	0.6	0.06	0
C’	0	0.13	0.83

Table 2. Matching demo

To gain the final result, the values in this matrix are compared to the duplicate threshold. The output of DUMAS is a list of pairs of duplicated columns.

5 Results

5.1 GUI

The GUI is a simple wrapper around the existing feature-set, written in Java’s Swing toolkit. The two tables are each displayed as taking the form of an “IntegratedTableModel”, a custom model which implements Java’s “AbstractTableModel” and can take two separate tables and display them combined. The main GUI elements are presented below.

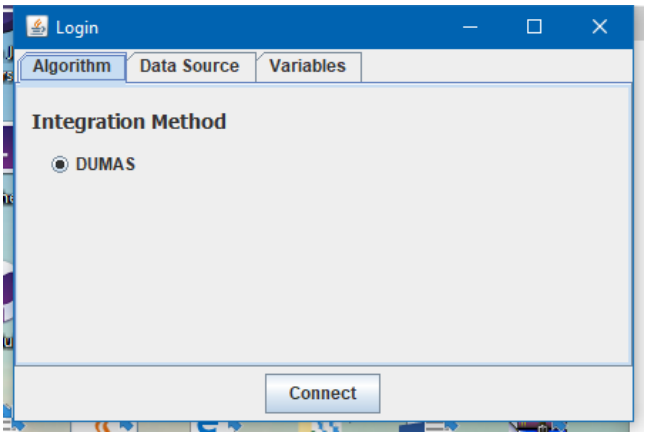

Figure 5. Integration algorithm selection

Figure 5 shows the initial settings window, which is opened as a dialog over the main application window. The first tab allows you to choose which integration back-end you will be using.

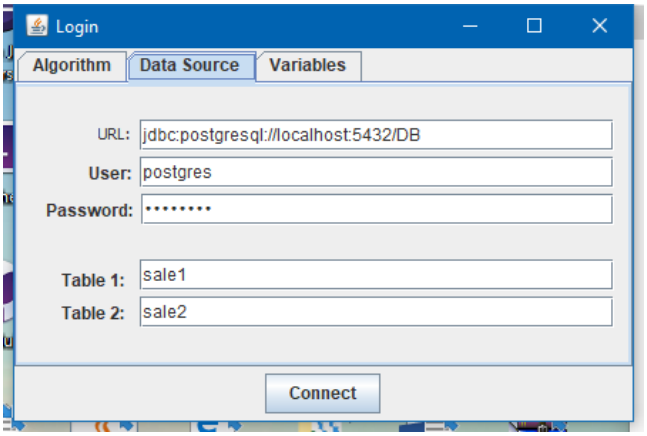

Figure 6. Data Source

Figure 6 shows the second tab of the settings window which allows you to set the connection and data source details. These values are pulled from the Config singleton instance, and when changed stay persistent for the entire session.

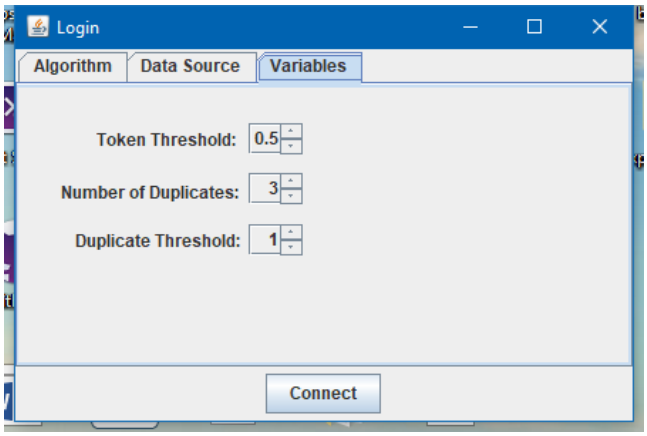

Figure 7. Algorithm variables

Figure 7 shows the final settings tab allow you to vary the variables used for the integration.

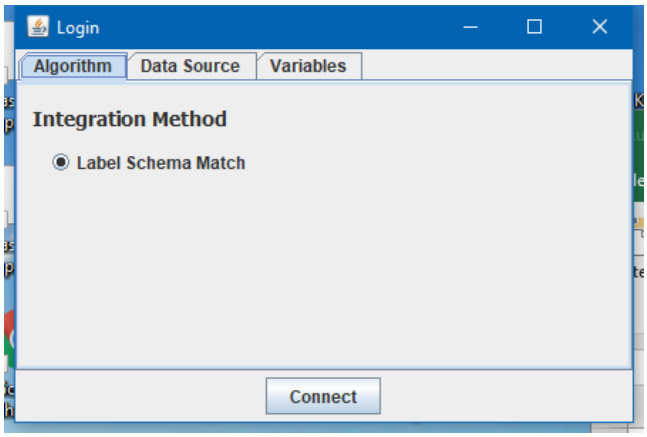

Figure 8. Integration algorithm selection

Figure 8 shows the login windows for label-based schema match.

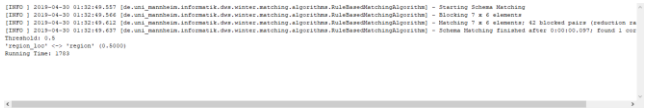

Figure 9. The result of label-based schema match

Figure 9 shows the results of label-based schema match when token threshold is 0.5.

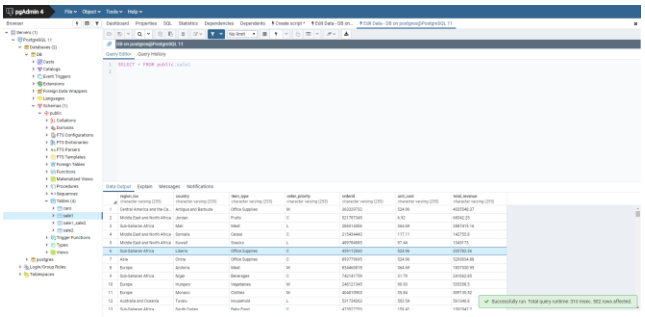

Figure 10. First source table in database

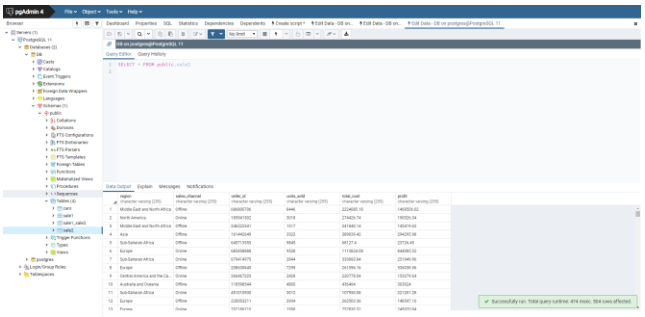

Figure 11. Second source table in database

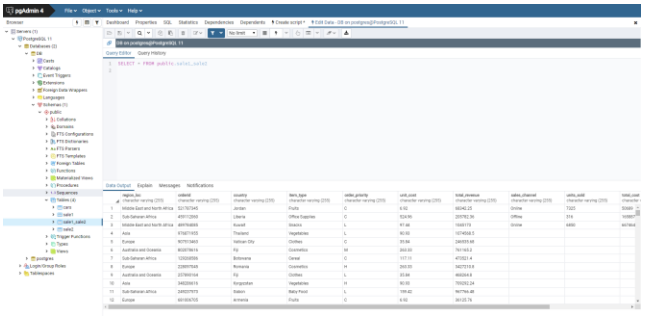

Figure 12. Second source table in database

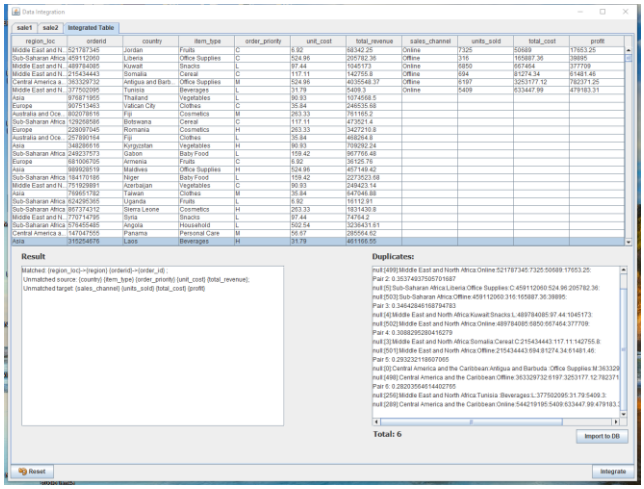

Figure 13. The integrated table

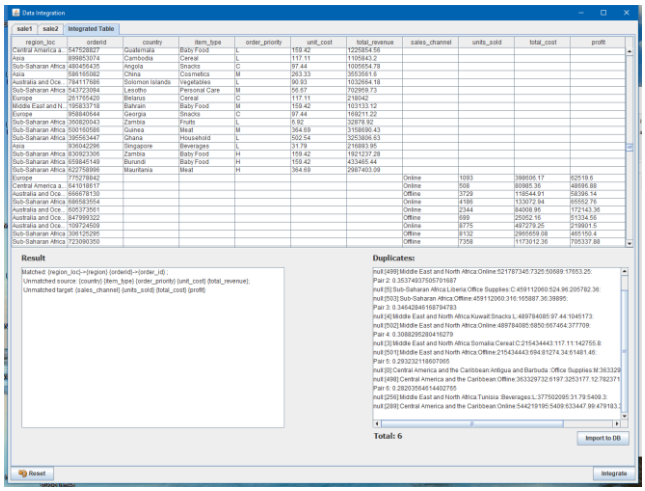

Figure 14. The unmatched table

Figure 10 shows the first source table in database.

Figure 11 shows the second source table in database.

Figure 12 shows the integrated table in database.

Figure 13 shows the integrated table after performing the integration.

Figure 14 shows the unmatched table after performing the integration.

5.2 Running time

After implemented DUMAS, we evaluated the performance by executing time. We also used label-based schema matching algorithm a completed application of which downloaded from the open source website. The comparison of two data integration algorithms based on the different token threshold was illustrated in Figure 15.

Token Threshold	Duplicate Threshold	DUMAS(milliseconds)	Label(milliseconds)
0.5	1	53.13435	25
0.6	1	40.098772	24
0.7	1	36.017014	23
0.8	1	44.429336	26
0.9	1	41.459948	24
1	1	44.743617	22

Figure 15. Comparison of DUMAS and Label based schema

6 Conclusions and future work

Obviously, Figure 15 shows the algorithm we implemented (DUMAS) is not better than the label-based schema match (WInte.r framework) in running-time. There are several reasons for this:

1. The DUMAS algorithm we implemented has not been further optimized. Due to time, we did not focus on the time complexity and simplicity of the code;

2. The two databases we use do not have a large amount of data, as discussed in Chapter 2 Related work. The algorithm advantage of DUMAS is very effective for schema alignment in databases based on a mere handful of duplicate tuples out of a very large dataset.

3. WInte.r framework is born in recent years, and DUMAS was proposed in 2005. The development of data integration is very fast, and the better performance of the label-based schema matching algorithm is beyond doubt.

We believe that by optimizing the time complexity of the system, finding and deleting redundant parts of the code, and increasing the amount of data in the database will improve the performance of DUMAS. We have a deep understanding of the purpose and significance of data integration, research background, implementation process and algorithm basis, which can lay the foundation for future study.

REFERENCES

[1] Lenzerini, Maurizio. "Data integration: A theoretical perspective." Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2002.

[2] Niswonger, L. H. R. M. B., et al. "Transforming heterogeneous data with database middleware: Beyond integration." Data Engineering 31 (1999).

[3] Doan, AnHai, Alon Halevy, and Zachary Ives. Principles of data integration. Elsevier, 2012.

[4] Doan, AnHai, Alon Halevy, and Zachary Ives. Principles of data integration. Elsevier, 2012. pp 9 – 10

[5] Bilke, Alexander, and Felix Naumann. "Schema matching using duplicates." 21st International Conference on Data Engineering (ICDE'05). IEEE, 2005.Sdfsdf

[6] Lehmberg, O., Brinkmann, A., & Bizer, C. WInte. r - A Web Data Integration Framework. ISWC 2017.

[7] Bilenko, Mikhail, and Raymond J. Mooney. "Adaptive duplicate detection using learnable string similarity measures." Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003.

[8] https://www.postgresql.org/files/documentation/pdf/11/postgresql-11-US.pdf

[9] Rajaraman, Anand, and Jeffrey David Ullman. Mining of massive datasets. Cambridge University Press, 2011.

[10] Cohen, William W., Pradeep Ravikumar, and Stephen E. Fienberg. "A Comparison of String Distance Metrics for Name-Matching Tasks." IIWeb. Vol. 2003. 2003.