HUMBOLDT-UNIVERSITÄT ZU BERLIN

Diploma Thesis Expose

Merging ETL Processes

Karsten Draba
kdraba@informatik.hu-berlin.de

Supervisors:

Prof. Dr. Ulf Leser
Humboldt Universität zu Berlin

Prof. Dr. Felix Naumann
Hasso-Plattner-Institute at the University of Potsdam

Adviser:

Alexander Albrecht
Hasso-Plattner-Institute at the University of Potsdam

submitted:
Berlin, on 20th May 2008

Faculty of Mathematics and Natural Sciences II
Department of Computer Science
Knowledge Management in Bioinformatics

Introduction

The specific processes and workflows of an organization are as much a part of their economic and functional basis, as the data these processes work on. But in contrast to data integration, the integration of processes has not gained much attention in information science research so far. This thesis proposes and develops the Merge operator as a fundamental operator for the integration of multiple processes. The focus is on data integration processes, commonly designed by use of ETL tools and applied in data warehouse environments.

Data integration processes can be described as sequences of activities transforming heterogenous data into a unified (integrated) representation. In the data warehouse context, these processes are called ETL processes, as they imply the extraction, transformation and loading of data. With the broad acceptance and usage of ETL tools, these ETL processes are given in a form that makes them available not only for (semi-) automatic execution, but also for further processing and optimization. The optimization of ETL processes is recently getting some attention within the database community (e.g. [1, 2]) and can greatly benefit from the research on query optimization for relational databases. Some techniques for relational databases can even be applied directly to ETL processes, due to the fact that some ETL activities can be expressed directly by use of relational algebra [3]. But a direct translation from ETL processes to relational algebra expressions is only possible in minor cases [1]. Nevertheless, the commonalities between the query optimization problem and the problem of ETL process optimization suggest the transfer of rule and cost based approaches for query optimization to the ETL process optimization problem.

So far only single ETL processes have been considered for optimization. The Merge operator goes beyond that, by aiming at the consolidation of multiple, often independent designed, ETL processes. Such a consolidation promises a gain in performance for the merged process, in contrast to the single processes, as common sub-processes can be exploited and data of a common domain can be processed together. Furthermore the merged process, as a single process, is open for further optimization using approaches for single process optimization. Equally important, the consolidated process represents a unified view of the merged processes and supports the definition of components and the reuse of sub-processes by the identification of common sub-processes.

The importance of a global optimization of ETL processes gets even clearer if one takes into account that ETL processes generally do not only access different, physically distributed data sources, but also execute their activities on different, physically distributed systems. In contrast to a batch of queries in traditional relational database systems, a batch of ETL processes can acquire many different resources for equal tasks, e.g. each ETL process in a batch may acquire its own local database to store intermediate results. Merging a batch of ETL processes therefore allows a better control and exploitation of the resources acquired by the batch, as common and functional equivalent resources are identified by common sub-processes.

Besides the semantic of the Merge operator, the following aspects of the thesis are introduced in the following sections of the expose.

- a systematic description of ETL processes supporting the definition of transformation rules
- the definition of common equivalent sub-processes and their identification via transformation rules
- the merge of multiple processes based on the identified common equivalent sub-processes

The expose concludes with a short remark to the implementation and a description of optional topics, not to be necessarily included in the final thesis.

1 Merge semantic

The aim of the Merge operator is to consolidate two or more processes. To ensure that the Merge operator does not change the semantics of the original processes, the intensions and extensions of all data sources and data targets involved in the process, have to be the same, whether the merged process has been executed or the batch of processes to merge. Furthermore, the data lineage for all dates has to be the same, whether the merged process or the batch has been executed. Besides lineage tracing through merged processes, this allows the correct application of operators like Invert on merged sub-processes. While lineage tracing is not actually implemented for the Merge operator in the context of this thesis, enforcing the correct lineage stresses the fact that, according to the proposed semantic, the Merge operator is consolidating only processes and not data in any way.

To merge processes, common sub-processes have to be identified. These processes can occur at the beginning, the end or in the middle of the processes to merge. This results in different patterns for merged processes, as shown in figure 1. (Some of these patterns are identified as frequent ETL process pattern in [4].) Naturally the common sub-processes can occur at different locations within different processes. And neither has a common sub-process to be shared by all processes, nor is the number of common sub-processes per process restricted to one.

2 Description of ETL processes

The possibility to merge processes, designed independently of each other, requires a common level of description. The architecture graph for ETL processes, proposed in [5], is used as a starting point for such a description. But to allow for an easy definition of transformation rules, some of the informations implicitly contained in the functionality description of activities have to be made explicit. Some of these informations are the functionality schema, the generated schema and the projected-out schema as suggested in [1]. By defining transformation rules and applying them to ETL processes the optimization problem can be transformed into a state-space based search problem [1]. The same holds true for the problem of finding equivalent sub-processes, where transformations can be used to span the search space too.

In the context of this thesis a selection of typical ETL stages is taken into account. All other stages are considered to be black boxes. Therefore it is assumed that an ETL process only consists of known stages and black boxes. While transformation rules for known ETL stages are defined, this is not possible for black boxes. Although some transformation rules for the relational algebra are used, new transformation rules are introduced to account for stages that are not expressible via relational algebra.

3 Equivalent sub-processes and search space generation

Two processes are considered to share a common sub-process whenever they contain an identical sequence of equivalent stages. Stages are considered to be equivalent whenever they have identical in- and out-schemas and the same functionality. The identification of common sub-processes requires therefore mappings between the different stages and their schemas. The thesis presupposes the existence and identification of these mappings by taking the validity of the naming principle for granted [1]. This means that all attributes with the same name are known to be identical while all attributes with different names are known to be different. Even if these mappings can be created by the known (semi-) automatic techniques, creating these mappings goes far beyond the scope of this thesis as the number and complexity of possible mappings increases drastically with the number of processes, their stages and the complexity of their schemas.

The validity of the naming principle is also taken for granted in regard to the functionality of stages. Therefore it can be assumed that all equally named stages implement the same functionality, independent of their actual schema attributes. This assumption is especially important to account for stages which functionality nothing is known about, i.e. black boxes. If the validity of the naming principle is not taken for granted, again a mapping has to be created. But as with schemas, creating such a mapping goes beyond the scope of this thesis. Creating such a mapping in a fully automatic fashion will not work in most of the cases. However, developing a sampling based semi-automatic approach may provide a good starting point for further research.

The smallest possible common sub-process of two or more ETL processes consists of a single common ETL stage. Starting from these common stages, transformation rules are used to create equivalent processes having sequences of equivalent stages in common. If these common stages are in the same order, a common sub-process is identified (see the example in figure 2).

The problem of common sub-expression identification is also addressed by research on query containment and multiple query optimization in relational databases. But while these provide a good starting point, there is a significant difference between the aims of these approaches and the one put forward in this thesis. Multiple query optimization is mainly interested in the identification, creation and exploitation of sharable intermediate results. In regard to the patterns shown in figure 1, this means that multiple query optimization is only interested in sub-processes at the beginning, i.e. sub-processes sharing the same data sources. This can be clearly seen on the table signatures, proposed for sub-process identification in [6], that mainly consist of the sources involved in a query. Therefore the main optimization question is, whether an identified common intermediate result should be exploited by materialisation [7], pipelining [8] or not at all.

In contrast, the identification of common sub-processes in ETL processes, as proposed here, is concerned with the identification of all common sequences of activities, whether they share data sources or not. This is due to the fact that on a higher abstraction level, e.g. the design, maintenance or administration level, recurring stage patterns are of a particular interest for re-use and resource management. Nevertheless, ETL process optimization and the MERGE operator can greatly benefit from multiple query optimization research, whenever common sub-processes with shared data sources are identified.

4 Merging ETL processes

The identified common sub-processes are the basis for the actual merge of the processes. One problem that arises at this point is to choose the appropriate sub-processes. Different equivalent representations of different processes can have different sub-process in common that exclude one another. Another problem is that not all identified common sub-processes may be promising candidates for a merge. For an example consider a projection on a specific set of attributes. Such a projection may occur in many processes. But as a projection is a simple, low cost and semantically not very significant operation, a merge based on a single common projection may not be of much use in most cases. A way to address these problems is to choose only the longest common sub-processes of a set of excluding sub-processes and to choose only sub-processes with a minimum length, or at least containing one stage of a specified set of stages. More generally, the MERGE operator has to support some kind of cost model, enabling the user to set up the sub-process selection according to her needs. This cost model has to be much broader than cost models in traditional or even distributed database systems, as it has to include different aspects of resource management too.

When actually merging the processes, new stages have to be inserted and existing stages have to be adapted to ensure the correct semantic of the MERGE operator and to uphold the semantics of the original processes. In the merged process, data processed by a common sub-process may originate from different original processes and therefore from different sources. To avoid an amalgamation and to ensure the correct forwarding of the data, its lineage may have to be traced. How exactly the common sub-processes have to be altered and integrated into the merged process, depends on the stages the sub-process consists of and its location in the original processes (see figure 1).

If the common sub-process is located at the beginning of the processes to merge, i.e. the processes have common sources, the merge of the sub-processes is simply the sub-process itself and its output can be simply forwarded to the sub-processes not to be merged (figure 3). This is the case multiple query optimization is based on and merging such processes can bring a direct performance gain. But if the sub-process is located in the middle or at the end of the processes to merge, a stage annotating the data with its lineage may have to be inserted (for annotation based lineage tracing in relational database systems see [9]). The lineage information is needed to control the data processing within the common sub-process and to divide the data at its end if necessary (figure 4, 5, 6, 7).

The disjunction of the data streams by their data lineage, in cases where the merged sub-process operates on data from different sources, may question the benefit of the MERGE operator in these cases. Processing the different data streams apart from each other within one process provides no performance gain if compared to the separate execution of the original processes. Therefore, it is important to stress that in these cases an increase in performance is not the overall goal of a merge, although it is possible in some cases (see figure 8). Rather, the benefits can be found in the optimization of resources, administration and maintenance and in re-usability (figure 9).

5 Implementation

The MERGE operator will be implemented in Java. An ETL process execution platform is provided by the Clover.ETL framework [10].

6 Optional and further research topics

To accentuate the core topics of the thesis, this sections lists some topics that are without a question connected to the realization of the MERGE, but may not be addressed in depth within the thesis, due to lack of time and space.

- Consider different MERGE semantics. For example the semantic of a MERGE that is not just consolidating processes but also integrating the data of these processes. In this case, data in common sub-processes is not to be divided. Rather, data with the same schema and processed by the same sub-process is assumed to be semantically equivalent and complementing one another. The extensions of the data targets are therefore not only dependent on the data sources they where in the original processes but on all data sources contributing to the common sub-process.

- A (semi-) automatic schema mapping technique is needed to map schemas between different processes as well as in a single process. Such a technique should not create all possible combinations of mappings as some of the already established mappings can be propagated through the process if mappings between the input and the output of stages are known.
- (Semi-) automatic techniques for mapping ETL stages are needed. Besides techniques based on stage names or ontologies, sampling based techniques seem to be promising. If the different stages are sampled by appropriate data sets, it will be possible to make statements about the probability of their functional equality.
- Furthermore, techniques are needed that allow an easy addition of new stages without making it necessary to define new transformation rules. Object oriented techniques, as of course used in the implementation of the MERGE operator, are one way to achieve this. But other techniques should be considered too. If, for example, new stages are described by already known stages, the known transformation rules can also be used for these new stages.
- Another important point is the optimization of the merged process. Although the MERGE operator has to support some kind of cost model, the implemented cost model will only be sufficient to decide which common sub-processes to use. Clearly, better cost models can be found and there may be much unused optimization potential after the merge. But as the cost model will typically have more dimensions than in traditional relational databases (e.g. administration and maintenance cost, data transfer volume, connection cost) developing one is not a trivial task.

References

[1] A. Simitsis, P. Vassiliadis, and T. K. Sellis. Optimizing ETL Processes in Data Warehouses. *Proceedings of the 21st International Conference on Data Engineering*, pages 564–575, 2005.

[2] M. V., H. S., O. S., B. M., V. M., A. M., and T. K. An Approach to Optimize Data Processing in Business Processes. In C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, editors, *VLDB*, pages 615–626, 2007.

[3] S. Dessloch, M. A. Hernández, R. Wisnesky, A. Radwan, and J. Zhou. Orchid: Integrating Schema Mapping and ETL. *24th International Conference on Data Engineering, April 7-12, 2008, Cancún, México*, 2008.

[4] P. Vassiliadis, A. Karagiannis, V. Tziovara, and A. Simitsis. Towards a Benchmark for ETL Workflows. In V. Ganti and F. Naumann, editors, *QDB*, pages 49–60, 2007.

[5] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. *Inf. Syst.*, 30:492–525, 2005.

[6] J. Z., P.-A. Larson, J. C. Freytag, and W. Lehner. Efficient exploitation of similar subexpressions for query processing. In C. Y. Chan, B. C. Ooi, and A. Zhou, editors, *SIGMOD Conference*, pages 533–544, 2007.

[7] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and Extensible Algorithms for Multi Query Optimization. In *SIGMOD Conference*, pages 249–260, 2000.

[8] N. N. Dalvi, S. K. Sanghai, P. Roy, and S. Sudarshan. Pipelining in Multi-Query Optimization. In *PODS*, 2001.

[9] D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya. An Annotation Management System for Relational Databases. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *VLDB*, pages 900–911, 2004.

[10] Clover.ETL. http://www.cloveretl.org/.

Appendix

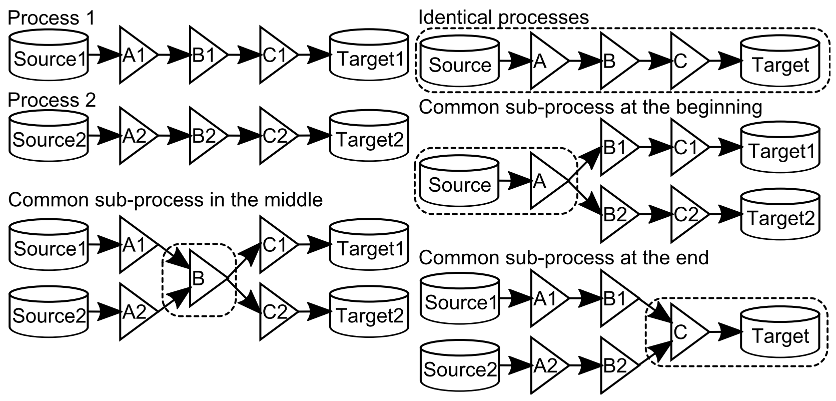

Figure 1: Different process patterns as they appear after a successful merge using common sub-processes at different locations in the processes to merge. In real processes mixtures of these patterns appear.

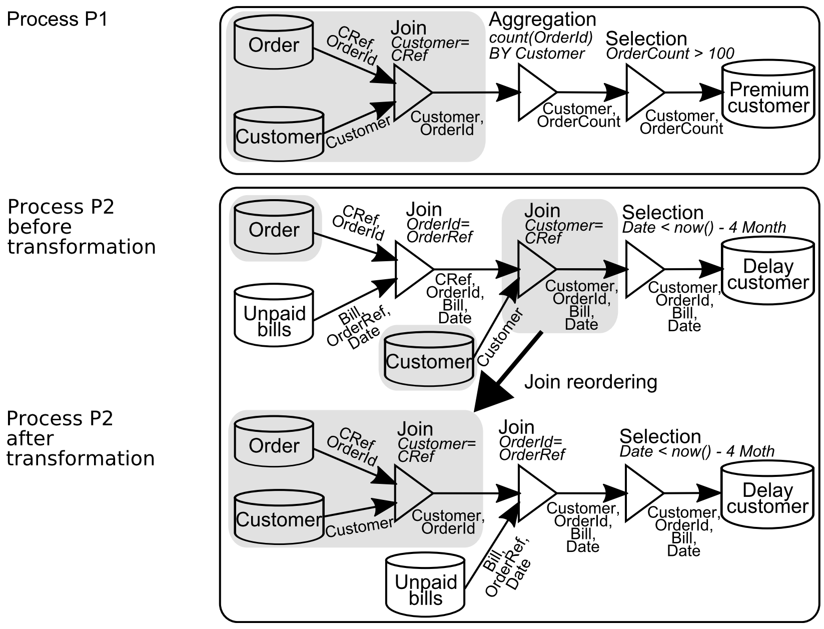

Figure 2: The figure shows two processes. The upper process identifies premium customers, i.e. customers with more than 100 orders, while the lower process identifies customers that are in a delay of payment, i.e. customers with unpaid bills older than four months. First, the two processes only have two sources, *Customer* and *Orders*, in common (the joins differ in their schemas). By reordering the joins in the lower process, a larger common sub-process emerges consisting of the two sources and a join. The merge of the two processes can be found in figure 3.

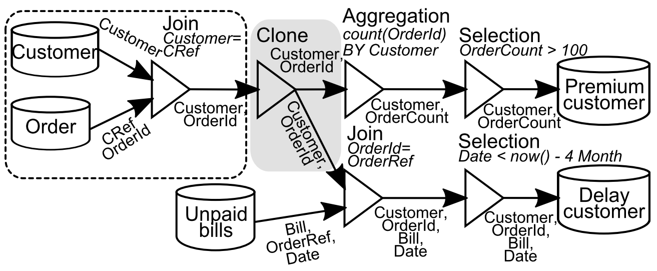

Figure 3: The figure shows the merge of the two processes depicted in figure 2. The grey highlighted sub-process is created by the MERGE operator, while the dashed sub-process is the common sub-process of the original processes. To forward the common processed data to the distinct sub-processes a CLONE operator, duplicating the data stream, has been inserted.

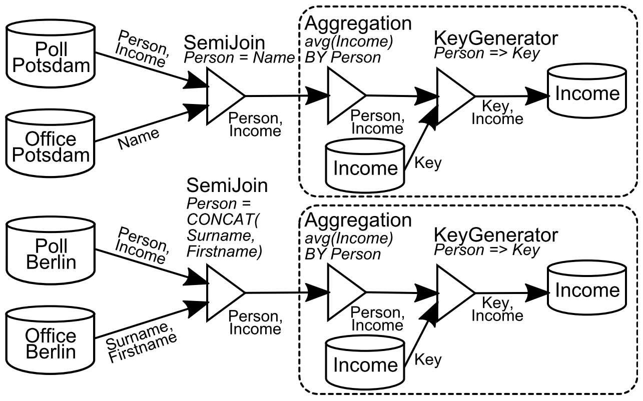

Figure 4: Assume a poll in different towns, surveying the incomes of their citizens. The two processes make anonymous the income of the different citizens and load it into the same data target. The existence of each citizen is checked by an appropriate town office. This way it is made sure that each person is uniquely identified within a town. But a person can not be uniquely identified among different towns. After checking her existence, the average income of a person is calculated and a new key is assigned to her, based on the already existing keys. Both processes differ in their sources as well as in the JOIN used to check for the existence of a citizen.

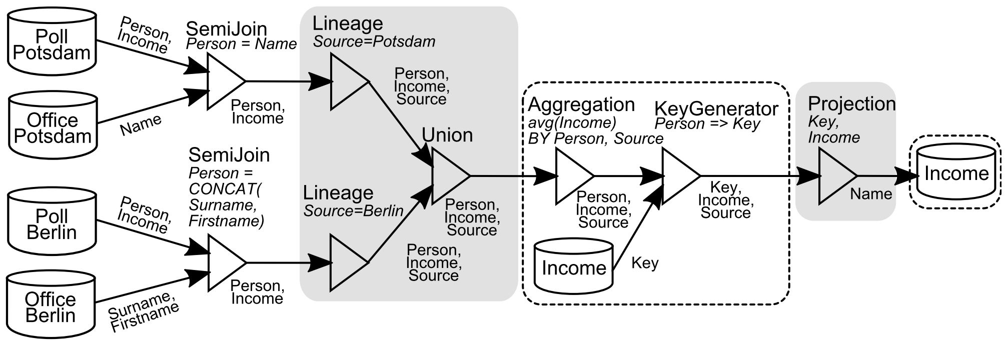

Figure 5: Here, the merge of the two processes shown in figure 4 is depicted. The inserted LINEAGE operator accumulates the data with its lineage, while the UNION operator pools them together for common processing. The accumulation with lineage is necessary, because persons are not uniquely identified among different sources. At the end of the common sub-process a PROJECTION is inserted to remove the lineage information from the data, before loading it into the data target. Note that the AGGREGATION properties had to be changed to prevent data from different sources to be aggregated. Also the schemas of the common sub-process had to be altered to account for the added lineage information.

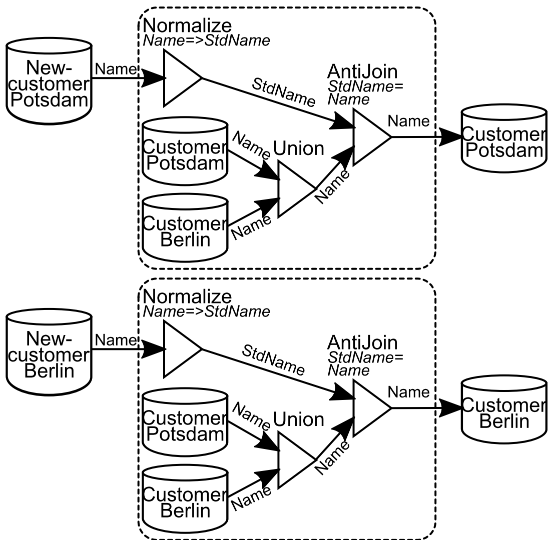

Figure 6: The two processes depicted here, first normalize the names of new customers and afterwards check whether a customer exists in a known customer database or not. If the new customer is not present in one of the known databases, it is added to the own customer database. Both processes differ in their sources as well as in their targets. Therefore the common sub-process is located in the middle.

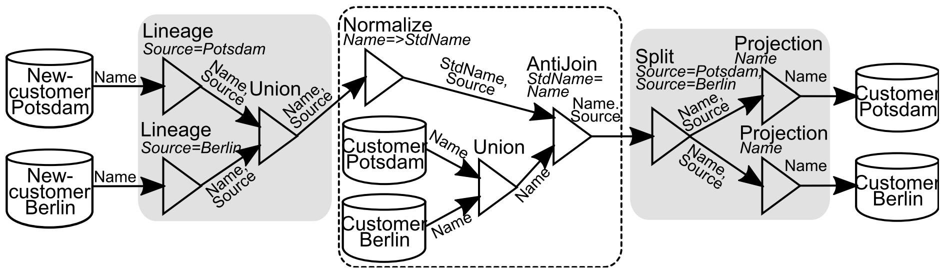

Figure 7: To merge the two processes shown in figure 6 it is necessary to accumulate the data about new customers with its lineage. After the common processing of the data, the lineage information is used to split and forward the processing results accordingly. Before actually loading the results, the lineage information has to be removed. In this example, only the schemas of the common sub-process have to be adjusted to account for the data lineage.

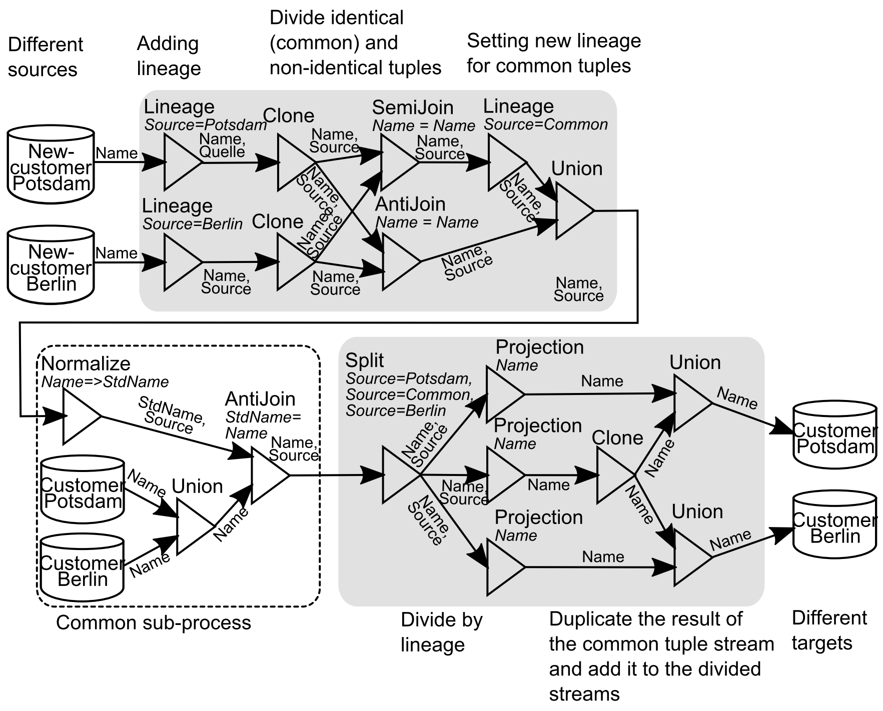

Figure 8: This figure shows again the merge of the two processes depicted in figure 6. This time, in contrast to figure 7, the merge aims at an increase in performance. To achieve this, all common new customers are identified using a SEMI-JOIN. This way, common new customers have to be processed only once. At the end of the common sub-process, the common new customers have to be duplicated and forwarded towards the non common sub-processes, i.e. the different sources. Obviously, this kind of merge is restricted to special cases. For instance there must not be any aggregation within the sub-process and duplicates within a source must not have any influence on the result. Furthermore, an increase in performance can only be achieved if the number of common tuples and the cost for processing them multiple times is so high that the cost for identifying the duplicates is exceeded.

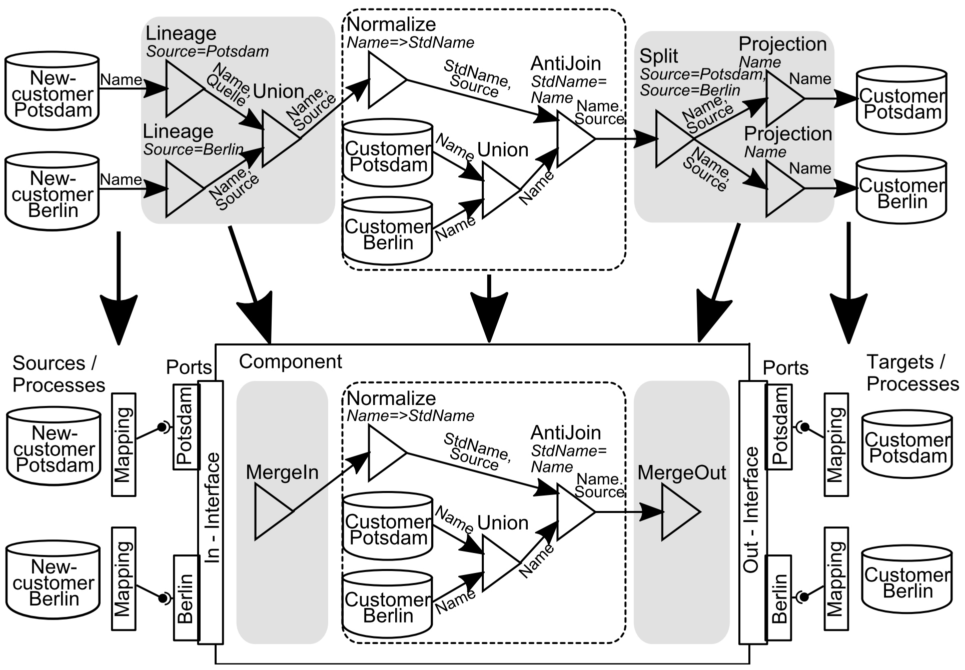

Figure 9: This figure shows how the merged process in figure 7 can be modeled as a component. While component specification itself is not part of this thesis, the figure exemplifies how the Merge operator can support re-usability and resource management, especially when merging processes that have no common sources. The common sub-process is embedded in Merge stages providing the necessary interfaces to embed this component into other ETL processes. This way the component can be considered as just another stage. Other processes can be connected to the component using ports, reflecting the data linieage used by the merged process. The important aspect of this component is that all processes using this component are using the same resources. To get an idea of the advantages of this approach, lets consider the case where each process requests its own resources. On the one hand this surely may improve the processing speed, but on the other hand requiring multiple resources may not only significantly increase cost but also bring with it an additional overhead in administering and maintaining these resources. But sharing resources can have its drawbacks too. Conflicts may arise and make a sequential execution of processes inevitable. The Merge operator allows to circumvent this problem by exploiting the data lineage. If for example one of the processes using the shared component blocks, this does not necessarily block the whole component. Because the data flows are independent of each other, due to their lineage, the resources of the blocked process (e.g. processing time, bandwidth, disk space etc.) can now be used by the other processes until the blocked process continues. Of course, this behavior can not be guaranteed in all cases and depends on the degree of independence of the merged processes. But as the processes get more dependent on one another, the possibilities rise for an increase in performance using the Merge operator.