A Survey of Extract–Transform–Load Technology

Panos Vassiliadis, University of Ioannina, Greece

ABSTRACT

The software processes that facilitate the original loading and the periodic refreshment of the data warehouse contents are commonly known as Extraction-Transformation-Loading (ETL) processes. The intention of this survey is to present the research work in the field of ETL technology in a structured way. To this end, we organize the coverage of the field as follows: (a) first, we cover the conceptual and logical modeling of ETL processes, along with some design methods, (b) we visit each stage of the E-T-L triplet, and examine problems that fall within each of these stages, (c) we discuss problems that pertain to the entirety of an ETL process, and, (d) we review some research prototypes of academic origin. [Article copies are available for purchase from InfoSci-on-Demand.com]

Keywords: *ETL, data warehouse refreshment*

INTRODUCTION

A data warehouse typically collects data from several operational or external systems (also known as the *sources* of the data warehouse) in order to provide its end-users with access to integrated and manageable information. In practice, this task of data collection (also known as *data warehouse population*) has to overcome several inherent problems, which can be shortly summarized as follows. First, since the different sources structure information in completely different schemata the need to transform the incoming source data to a common, "global" data warehouse schema that will eventually be used by end user applications for querying is imperative. Second, the data coming from the operational sources suffer from quality problems, ranging from simple misspellings in textual attributes to value inconsistencies, database constraint violations and conflicting or missing information; consequently, this kind of "noise" from the data must be removed so that end-users are provided data that are as clean, complete and truthful as possible. Third, since the information is constantly updated in the production systems that populate the warehouse, it is necessary to refresh the data warehouse contents regularly, in order to provide the users with up-to-date information. All these problems require that the respective software processes are constructed by the data warehouse development team (either manually, or via specialized tools) and executed in appropriate time intervals

for the correct and complete population of the data warehouse.

The software processes that facilitate the population of the data warehouse are commonly known as Extraction-Transformation-Loading (ETL) processes. ETL processes are responsible for (i) the extraction of the appropriate data from the sources, (ii) their transportation to a special-purpose area of the data warehouse where they will be processed, (iii) the transformation of the source data and the computation of new values (and, possibly records) in order to obey the structure of the data warehouse relation to which they are targeted, (iv) the isolation and cleansing of problematic tuples, in order to guarantee that business rules and database constraints are respected and (v) the loading of the cleansed, transformed data to the appropriate relation in the warehouse, along with the refreshment of its accompanying indexes and materialized views.

A naïve, exemplary ETL scenario implemented in MS SQL Server Integration Services is depicted in Figure 1. The scenario is organized in two parts. The first part, named *Extraction task* (Figure 1a), is responsible for the identification of the new and the updated rows in the source table *LINEITEM*. The idea is that we have an older snapshot for line items, stored in table *LINEITEM* which is compared to the new snapshot coming from the sources of the warehouse (depicted as *NEW_LINEITEM*) in Figure 1b. Depending on whether a row is (a) newly inserted, or, (b) an existing tuple that has been updated, it is routed to the table *LINEITEM*, via the appropriate transformation (remember that insertions and updates cannot be uniformly handled by the DBMS). Once the table *LINEITEM* is populated, the second part of the scenario, named *Transform & Load task* (Figure 1a) is executed. This task is depicted in Figure 1c and its purpose is to populate with the update information several tables in the warehouse that act as materialized views. The scenario first computes the value for the attribute *Profit* for each tuple and then sends the transformed rows towards four "materialized" views that compute the following aggregated measures (keep in mind that *ExtendedPrice* refers to the money clients pay per line item, *PartKey* is the primary key for items and *SuppKey* is the primary key for suppliers):

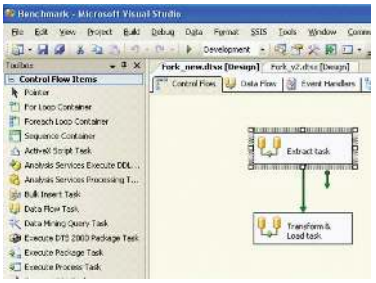
- *View A*: aggregate profit and average discount grouped by *PartKey* and *SuppKey*
- *View B*: average profit and extended price grouped by *PartKey* and *LineStatus*
- *View C*: aggregate profit and extended price grouped by *LineStatus* and *PartKey*
- *View D*: aggregate profit and extended price grouped by *LineStatus*

As one can observe, an ETL process is the synthesis of individual tasks that perform extraction, transformation, cleaning or loading of data in an execution graph – also referred to as a workflow. Also, due to the nature of the design artifact and the user interface of ETL tools, an ETL process is accompanied by a plan that is to be executed. For these reasons, in the rest of this survey we will use the terms *ETL process*, *ETL workflow* and *ETL scenario* interchangeably.

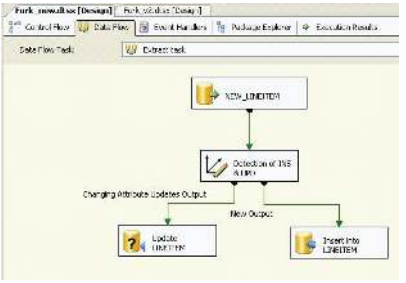
The historical background for ETL processes goes all the way back to the birth of information processing software. Software for transforming and filtering information from one (structured, semi-structured, or even unstructured) file to another has been constructed since the early years of data banks, where the relational model and declarative database querying were not invented. Data and software were considered an inseparable duo for data management by that time and thus, this software was not treated as a stand-alone, special purpose module of the information system's architecture. As Vassiliadis and Simitsis (2009) mention "since then, any kind of data processing software that reshapes or filters records, calculates new values, and populates another data store than the original one is a form of an ETL program."

After the relational model had been born and the declarative nature of relational database querying had started to gain ground, it was quite natural that the research community would try to apply the declarative paradigm to data

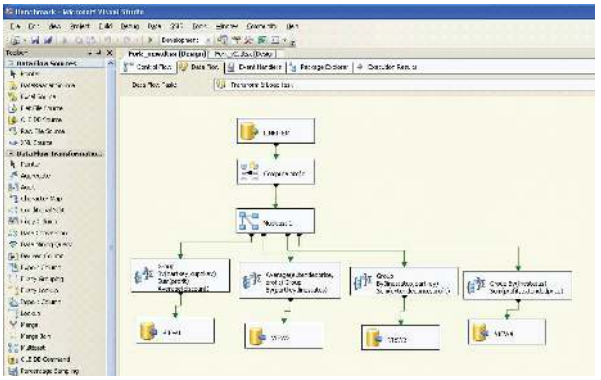
Figure 1. The environment of Extraction-Transformation-Loading processes



(a) Control flow of an ETL scenario



(b) Simple extraction part of an ETL scenario



(c) Computation of extra values and multiple aggregations as part of an ETL scenario

transformations. The EXPRESS system (Shu, Housel, Taylor, Ghosh, & Lum, 1977) is the first attempt that we know with the purpose of producing data transformations, taking as input data definitions or the involved nonprocedural statements. During the later years, the emphasis on the data integration problem was significant, and wrapper-based exchange of data between integrated database systems was the closest thing to ETL that we can report – for example, see Roth and Schwarz (1997).

ETL has taken its name and existence as a separate set of tools and processes in the early '00s. Despite the fact that data warehouses had become an established practice in large organizations since the latest part of the '90s, it was only in the early '00s that both the industrial vendors and the research community cared to deal seriously with the field. It is noteworthy that till then, the research community had typically hidden the internals of ETL process "under the carpet" by treating the data warehouse as a set of materialized views over the sources. At the same time, the industrial vendors were focused on providing fast querying and reporting facilities to end users. Still, once data warehouses were established as a practice, it was time to focus on the tasks faced by the developers. As a result, during the '00s, the industrial field is flourishing with tools from both the major database vendors and specialized companies and, at the same time, the research community has abandoned the treatment of data warehouses as collections of materialized views and focuses on the actual issues of ETL processes.

The intention of this survey is to present the research work in the field of ETL in a structured way. The reader is assumed to be familiar with

the fundamental concepts of databases and data warehousing. Since the focus is on ETL processes, we will avoid the detailed coverage of research topics like materialized view refreshment and data cleaning that are close to ETL processes (in fact, in practical situations, these tasks can be important constituents of an ETL workflow) but have an existence of their own in the research literature. Therefore, the coverage of these topics mainly includes a discussion of the problems under investigation and refers the reader to dedicated surveys with a broad discussion of the state of the art.

The discussion will start with the examination of design issues and then, it will proceed to cover technical issues for ETL processes. In Section 2, we cover the conceptual and logical modeling of ETL processes, along with some design methods. In Section 3, we visit each stage of the E-T-L triplet, and examine problems that fall within each of these stages. Then, in Section 4, we discuss problems that pertain to the entirety of an ETL process (and not just in one of its components), such as issues around the optimization, resumption and benchmarking of ETL processes, along with a discussion of the newest trend in ETL technology, near-real time ETL. In Section 5, we review some research prototypes with academic origin. Finally, in Section 6, we conclude our coverage of the topic with an eye for the future.

STATE OF THE ART FOR THE DESIGN AND MODELING OF ETL PROCESSES

Traditionally, a large part of the literature, the research activity, and the research community of data warehouses is related to the area of conceptual modeling and design. To a large extent, this is due to the fact that data warehouse projects are highly costly and highly risky endeavors; therefore, careful design and preparation are necessary. Moreover, due to their complexity, data warehouse environments should be carefully documented for maintenance purposes. ETL processes could not escape the above rule and, therefore, they have attracted the attention of the research community. A typical reason for this attention is also the fact that mainstream industrial approaches –see for example Kimbal, Reeves, Ross & Thornthwaite (1998), or Kimball & Caserta (2004)– focus on the physical-level details and lack a principled approach towards the problem of designing a data warehouse refreshment process. In this section, we will discuss the appearance of research efforts for the conceptual modeling of ETL processes with a chronological perspective and also cover some efforts concerning the logical modeling and design methods for this task.

UML Meta Modeling for Data Warehouses

The first approaches that are related to the conceptual design of data warehouses were hidden in discussions pertaining to data warehouse metadata. Stöhr, Müller, & Rahm (1999) propose an UML-based metamodel for data warehouses that covers both the back stage and the front-end of the data warehouse. Concerning the back stage of the data warehouse, which is the part of the paper that falls in the scope of this survey, the authors cover the workflow from the sources towards the target data stores with entities like *Mapping* (among entities) and *Transformation* (further classified to aggregations, filters, etc.) that are used to trace the inter-concept relationships in this workflow environment. The overall approach is a coherent, UML-based framework for data warehouse metadata, defined at a high-level of abstraction. The main contribution of the authors is that they provide a framework where specialized ETL activities (e.g., aggregations, cleanings, pivots, etc) can be plugged in easily via some kind of specialization.

First Attempts towards a Conceptual Model

The first attempt towards a conceptual model dedicated to the design and documentation of the

data warehouse refreshment was presented by Vassiliadis, Simitsis & Skiadopoulos (DOLAP 2002). The main motivation for the model was the observation that during the earliest stages of the data warehouse design, the designer is concerned with the analysis of the structure and content of the existing data sources and their mapping to the common data warehouse model. Therefore, a formal model for this task was necessary at the time.

The model of Vassiliadis et al (DOLAP 2002) involves concepts (standing for source and warehouse data holders) and their attributes, which define their internal structure. Part-of relationships correlate concepts and their constituent attributes. Attributes of source and warehouse concepts are related to each other with provider relationships. If a transformation takes place during the population of a target attribute, then the provider relationship passes through a transformation node in the model. Multiple transformations are connected via serial composition relationships. The model allows the definition of ETL constraints that signify the need for certain checks at the data (e.g., a certain field must be not null or within a certain range value, etc). Finally, multiple source candidates for the population of a warehouse concept are also tracked via candidate relationships. This is particularly useful for the early stages of the design, where more than one sources can be chosen for the population of a warehouse fact table. If one of them is eventually chosen (e.g., due to its higher data quality), then this source concept is characterized as the active candidate for the model.

A second observation of Vassiliadis et al (DOLAP 2002) was that is practically impossible to forecast all the transformations and cleanings that a designer might ever need. So, instead of proposing a closed set of transformations, the authors discuss the extensibility of the model with template transformations that are defined by the designer.

UML Revisited for ETL Processes

Trujillo & Luján-Mora (2003) revisit the conceptual modeling of ETL workflows from the view point of UML with the basic argument that the previous modeling by Vassiliadis et al is done via an ad-hoc model. So, the authors try to facilitate the modeling effort for ETL workflows with standard methods and they employ UML for this purpose.

It is interesting that the authors employ class diagrams and not activity diagrams for their modeling. The participating entities are UML packages; this is a powerful feature of the model, since it allows the arbitrary nesting of tasks. This nesting mechanism, quite common to UML, alleviates the complexity of the model of Vassiliadis et al., since it allows a gradual zooming in and out of tasks at different levels of detail. The main reason for dealing with class diagrams is that the focus of the modeling is on the interconnection of activities and data stores and not on the actual sequence of steps that each activity performs. Under this prism, whenever an activity A_1 populates an activity A_2, then A_2 is connected to A_1 with a dependency association.

Then, Trujillo & Luján-Mora (2003) provide a short description for a set of commonly encountered activities. Each such template activity is graphically depicted as an icon. The activities covered by the authors are: aggregation, conversion, logging, filtering, join, and loading of data, as well as checks for incorrect data, merging of data coming from different sources, wrapping of various kinds of external data sources and surrogate key assignment.

The authors build their approach on a generic structure for the design process for ETL workflows. So, they structure their generic design process in six stages, specifically, (i) source selection, (ii) source data transformation, (iii) source data join, (iv) target selection, (v) attribute mappings between source and target data and (vi) data loading.

UML and Attribute Mappings

One of the main assumptions of the approach of Trujillo & Luján-Mora (2003) was that the user must not be overwhelmed with the multitude of attribute mappings between sources, ETL activities and target warehouse tables. Still, as already mentioned, such detail is important for the back-stage of the data warehouse. Without capturing the details of the inter-attribute mappings, important transformations, checks and contingency actions are not present in the documentation of the process and can be ignored at the construction/generation of code. Despite the effort needed, this documentation can be useful at the early stages of the project, where the designer is familiarized with the internal structure and contents of the sources (which include data quality problems, cryptic codes, conventions made by the administrators and the programmers of the sources and so on). Moreover, this documentation can also be useful during later stages of the project where the data schemata as well as the ETL tasks evolve and sensitive parts of the ETL workflow, both at the data and the activities, need to be highlighted and protected.

It is interesting that no standard formalism like the ER model or the UML treats attributes as *first-class citizens* –and as such, they are unable to participate in relationships. Therefore, Luján-Mora, Vassiliadis & Trujillo (2004) stress the need to devise a mechanism for capturing the relationships of attributes in a way that is (a) as standard as possible and (b) allows different levels of zooming, in order to avoid overloading the designer with the large amount of attribute relationships that are present in a data warehouse setting.

To this end, the authors devise a mechanism for capturing these relationships, via a UML *data mapping* diagram. UML is employed as a standard notation and its extensibility mechanism is exploited, in order to provide a standard model to the designers. Data mapping diagrams treat relations as classes (like the UML relational profile does). Attributes are represented via proxy classes, connected to the relation classes via stereotyped "Contain" relationships. Attributes can be related to each other via stereotyped "Map" relationships.

A particular point of emphasis made by Luján-Mora et al. (2004) is the requirement for multiple, complementary diagrams at different levels of detail. The authors propose four different layers of data mappings, specifically, (a) the database level, where the involved databases are represented as UML packages, (b) the dataflow level, where the relationships among source and target relations are captured, each in a single UML package, (c) the table level, where the dataflow diagram is zoomed in and each individual transformation is captured as a package, and (d) the attribute level, which offers a zoom-in to a table-level data mapping diagram, with all the attributes and the individual attribute level mappings captured.

State-of-the-Art at the Logical Level

Apart from the conceptual modeling process that constructs a first design of the ETL process, once the process has been implemented, there is a need to organize and document the meta-information around it. The organization of the metadata for the ETL process constitutes its *logical level* description – much like the system's catalog acts as the logical level description of a relational database.

Davidson & Kosky (1999) present WOL, a Horn-clause language, to specify transformations between complex types. The transformations are specified as rules in a Horn-clause language. An interesting idea behind this approach is that a transformation of an element can be decomposed to a set of rules for its elements, thus avoiding the difficulty of employing complex definitions.

As already mentioned, the first attempt towards a systematic description for the metadata of the ETL process go back to the works by Stöhr et al. (1999) and Vassiliadis, Quix, Vassiliou & Jarke (2001). This research has been complemented by the approach of Vassiliadis, Simitsis & Skiadopoulos (DMDW

2002) where a formal logical model for ETL process is proposed. The main idea around this model concerns the capturing of the data flow from the sources to the warehouse. This means that meta-information concerning relations or files and their schemata is kept, along with the meta-information on the activities involved and their semantics. Naturally, the interconnection of all these components in a workflow graph that implements the ETL process is also captured. The proposal of Vassiliadis et al (DMDW 2002) models the ETL workflow as a graph. The nodes of the graph are activities, recordsets (uniformly modeling files and relations) and attributes; note that recordsets have a schema in the typical way and activities have input schemata, as well as an output and a rejection schema (the latter serving the routing of offending rows to quarantine). Part-of edges connect the attributes with their encompassing nodes and provider relationships show how the values are propagated among schemata (in other words, provider relationships trace the dependencies in terms of data provision between one or more data provider attributes and a populated, data consumer attribute). Derived relationships also capture the transitive dependency of attributes that are populated with values that are computed via functions and parameters. An ETL workflow is a serializable combination of ETL activities, provider relationships and data stores. The overall modeling of the environment is called *Architecture Graph*; in other words, an architecture graph is the logical level description of the data flow of an ETL process.

The above mentioned model suffered from the lack of concrete modelling of the semantics of the activities as part of the graph. In other words, the provider relationships capture only the dependencies of a consumer attribute to its data providers, without incorporating the actual filterings and transformations that take place on the way from the provider to the consumer. This shortcoming was complemented by the work of Vassiliadis, Simitsis, Georgantas, Terrovitis, & Skiadopoulos (CAiSE 2003, IS 2005, DaWaK 2005, ER 2005). Due to the complicated nature of the internal semantics of the activities, the fundamental idea of these papers is to describe the meta-information via a series of intermediate schemata. Coarsely speaking, the semantics of the activity are related to this graph of internal, intermediate schemata via a simple convention that a schema corresponds to a predicate in rule-based language. LDL, a Datalog variant is the chosen language for this line of papers. Then, each rule of the form

```
OUTPUT<-INPUT, filters, functions,
input-to-output mappings
```

practically stands for a combination of inputs, outputs, comparison nodes and functions that connect the respective schemata via provider edges (see the long version of Vassiliadis et al, ER 2005 for a detailed description).

The works by Vassiliadis et al. (CAiSE 2003, IS 2005) also present a template language that allows ETL designers to define reusable templates of LDL programs via the appropriate macros (see also the discussion in the section "Systems" for the ARKTOS tool).

From the very beginning, this line of work was concerned with the exploitation of the meta-information for ETL processes. The papers by Vassiliadis et al. (DMDW 2002; ER 2005) are concerned with metrics for the identification of important properties of an ETL design. The metrics proposed by Vassiliadis et al. (DMDW 2002) are simple but quite powerful and capture the degree of dependence of a node to other nodes and vice versa. The metrics proposed by Vassiliadis et al. (ER 2005) are based on a rigorous framework for metrics of graph-based software constructs and show the size, cohesion, coupling, and complexity of a constructed ETL process.

Semantics-aware Design Methods for ETL

A semi-automatic transition from the conceptual to the logical model for ETL processes has been proposed first by Simitsis (2005) and later by Simitsis & Vassiliadis (DSS 2008). Simple mappings involve the mapping of concepts to

relations, and the mapping of transformations and constraint checks to the respective activities. The hardest problem that is dealt with is the mapping of the declarative requirements for transformations and checks at the conceptual level to a sequence of activities at the logical level. The paper proposes an algorithm that groups transformations and checks in *stages*, with each stage being a set of activities whose execution order can be transposed. Stage derivation is based on the idea of dependency: if the input attributes of a certain activity *a* depend on the (output) attributes of another activity or recordset *b*, then *a* should be on a higher order stage than *b*. Binary activities and persistent record sets typically act as boundaries for stages. Since the presented algorithms start with simple cases of single source – single target pairs, binary activities used as pivotal points of the scenario for the derivation of common sub-paths (after the output of the binary activity). Overall, since the ordering of stages is quite simple, the determination of the execution orders for all the activities is greatly simplified.

Skoutas & Simitsis (DOLAP 2006, IJSWIS 2007) use ontologies to construct the conceptual model of an ETL process. Based on domain knowledge of the designer and user requirements about the data sources and the data warehouse, an appropriate OWL ontology is constructed and used to annotate the schemas of these data stores. Then, according to the ontology and these annotations, an OWL reasoner is employed to infer correspondences and conflicts between the sources and the target, and to propose conceptual ETL operations for transferring data between them. Skoutas and Simitsis (NLDB 2007) use ontologies to document the requirements of an ETL process. The proposed method takes a formal, OWL description of the semantic descriptions of (a) the data sources, (b) the data warehouse, as well as (c) the conceptual specification of an ETL process and translates them to a textual format that resembles natural language. This is facilitated via a template-based approach for the constructs of the formal description.

INDIVIDUAL OPERATORS AND TASKS IN ETL SCENARIOS

In this section, we organize the review of the literature based on the constituents of the ETL process, specifically, the extraction, transformation (&cleaning), and loading phases. For each phase, we discuss practical problems and solutions proposed by the research community.

Research Efforts Concerning Data Extraction Tasks

The extraction is the hardest part of the refreshment of the data warehouse. This is due to two facts. First, the extraction software must incur minimum overheads to the source system both at runtime and at the nightly time window that is dedicated to the refreshment of the warehouse. Second, the extraction software must be installed at the source side with minimum effect to the source's software configuration. Typical techniques for the task include taking the difference of consecutive snapshots, the usage of timestamps (acting as transaction time) in source relations, or the "replaying" the source's log file at the warehouse side. Non-traditional, rarely used techniques require the modification of the source applications to inform the warehouse on the performed alterations at the source, or, the usage of triggers at the source side.

Snapshot difference between a newer and an older snapshot of a relation seems straightforward: In principle, the identification of records that are present in one snapshot and not in the other gives us the insertions and deletions performed; updates refer to two tuples that are present in the two snapshots and share the same primary key, but different non-key values. Despite this theoretical simplicity, performance considerations are very important and pose a research problem.

The research community has dealt with the problem from the mid '80s. Lindsay, Haas, Mohan, Pirahesh, & Wilms (1986) propose a timestamp based algorithm for detecting inser-

tions, deletions and updates in two snapshots. A simple algorithm annotating changes with timestamps and comparing the values for tuples changed after the last comparison is proposed. Improvements concerning the identification of "empty areas" in the table speed up the process further. Labio & Garcia-Molina (1996) have presented the state of the art method on the topic of snapshot difference. The paper assumes two snapshots as input and produces a sequence of insertion, deletion and update actions to tuples, with each tuple being identified by its key. Several algorithms are discussed in the paper, including sort-merge outerjoins and partitioned hash joins. These algorithms can be extended with compression techniques in order to reduce the size of the processed data and the incurred I/Os as well as in order to exploit the opportunity of fitting one of the two snapshots in main memory. Compression is applied to each tuple individually (or to a whole block) and interestingly, the employed compression function is orthogonal to the algorithm of choice, with the extra observation that lossy compression methods might introduce erroneously identified modifications with very small probability. Overall, the investigated compression techniques include two methods, (a) simple tuple compression and (b) simple tuple compression with a pointer to the original uncompressed record. Then, the authors propose several algorithms for the identification of modifications. The first algorithm performs a sort-merge outer-join. Due to the anticipated periodic execution of the algorithm it is safe to assume that the previous snapshot is sorted. The algorithm sorts the new snapshot into runs and proceeds as the typical sort-merge join with the extra fundamental checks that are mentioned above, concerning the characterization of a tuple as an insertion (deletion), update, or existing entry. A variant of the algorithm involves the usage of compressed snapshot. In this case, the pointer to the original tuple can help with the identification of updates. Another variant of the algorithm concerning the hash join is also discussed. The main proposal of the paper, though, is the so-called window algorithm, which exploits the idea that the tuples that are present in both snapshots are found in approximately the same place in the two snapshots. The algorithm uses an input buffer and a buffer for candidate modified tuples (*ageing buffer* in the paper's terminology) per snapshot. The algorithm fills the input buffers and compares their contents. Tuples found identical in the two buffers are no longer considered. Misses are tested over the ageing buffer of the other snapshot. If a match occurs, the tuples are further ignored. The tuples that remain in the input buffers after these tests are candidates to be insertions or deletions and they are pushed to the ageing buffer of their snapshot. Due to space requirements, if an ageing buffer is full, it must be emptied. To this end, a queue of pointers is maintained, keeping track of the order in which the tuples entered the buffer; if the buffer is full, the oldest tuples are emptied. The algorithm is quite efficient and safe if the same records are physically placed in nearby areas in the two snapshots and the experimental results have proved that this is a realistic assumption.

Research Efforts Concerning Data Transformation Tasks

Although naïve data transformations are inherently built inside SQL and relational algebra, transformations that are used in ETL scenarios have not really found their way in the research literature. The main reason for this is probably due to the fact that transformations of this kind are typically ad-hoc and rather straightforward if studied individually; the complexity arises when their combination in a workflow is introduced. Nevertheless, there are some approaches that try to deal with the way to transform input to output data efficiently, and elegantly in terms of semantics.

The Pivoting Problem

Pivoting refers to a common spreadsheet operation for the tabular presentation of data to the end user, which is also quite common in ETL processes. Assume that a user wants to repre-

sent information about employees' revenues, with the revenues of each employee split into categories like salary, tax, work bonus, and family bonus. There are two possible organizations for this kind of data. First, a row-oriented organization is based on a table with the above categories represented as attributes -- e.g., consider a relation of the form EMP_r(*EID*, *ESAL*, *ETAX*, *EWBonus*, *EFBonus*). A second, attribute-value representation splits the revenues of each employee into different rows – e.g., consider a relation of the form EMP_{av}(*EID*, *RID*, *Amount*) with *RID* being a foreign key to a table *REVENUE_CATEGORIES*(*RID*, *RDescr*) that contains values {(*10*, *Sal*), (*20*,*Tax*), …}. Pivoting is the operation that transforms relation EMP_{av} to EMP_r; unpivot is the inverse operation. Cunningham, Galindo-Legaria & Graefe (2004) discuss efficient ways to implement these two operations in a DBMS environment. The authors start by suggesting compact extensions of SQL to allow the end user to directly express the fact that this is the requested operation (as opposed to asking the user perform pivot/unpivot through complicated expressions that the optimizer will fail to recognize as one operation). Special treatment is taken for data collisions and NULL values. The authors build upon the fact that pivot is a special case of aggregation, whereas unpivot is a special case of a correlated nested loop self join and explore query processing and execution strategies, as well as query optimization transformations, including the possibilities of pushing projections and selections down in execution plans that directly involve them.

Data Mappers

A sequence of papers by Carreira et al. (DaWaK 2005, DKE 2007, ICEIS 2007) explore the possibilities imposed by data transformations that require one-to-many mappings, i.e., transformations that produce several output tuples for each input tuple. This kind of operations is typically encountered in ETL scenarios. Since relational algebra is not equipped with an operator that performs this kind of input-output mapping, Carreira et al. extend it by proposing a new operator called *data mapper* and explore its semantics and properties. In this discussion, we mainly focus on the work of Carreira et al (DKE, 2007) which is a long version of a previous work (DaWaK 2005). The authors define the data mapper operator as a computable function mapping the space of values of an input schema to the space of values of an output schema. Mappers are characterized as single or multi-value mappers, depending on whether one or more tuples occur at the output given an arbitrary tuple at the input of the operator. Mappers under investigation are minimalistic operators and should be highly cohesive (i.e., they should do exactly one job); to this end, a mapper is defined to be in normal form if it cannot be expressed as the composition of two other mappers. Carreira et al. propose algebraic rewritings to speed up the execution of composite expressions of the extended relation algebra. Specifically, a data mapper can be combined with a (originally subsequent) selection condition, if the condition operates on a parameter of a mapper function. A second rule directs how a selection condition that uses attributes of the output of a data mapper can be translated to the attributes that generate them and thus be pushed through the mapper. A third rule deals with how projection can help avoid unnecessary computations of mappers that will be subsequently projected-out later. Finally, Carreira et al (ICEIS 2007) report some first results on their experimentation with implementing the data mappers in a real RDBMS. A more detailed description of alternative implementations is given by Carreira et al (QDB 2007) concerning unions, recursive queries, table functions, stored procedures and pivoting operations as candidates for the implementation of the data mapper operator. The first four alternatives where used for experimentation in two different DBMSs and table functions appear to provide the highest throughput for data mappers.

The Data Lineage Problem

Apart from efficiently performing the transformations in an ETL process, it is also important to be able to deal with the inverse problem. Given a certain warehouse tuple (or, set of tuples), the *data lineage problems* involves the identification of the input data that are the originators of the given tuple (or, tuples).

Cui and Widom have investigated the problem of data lineage in a set of papers (Cui & Widom, 2001 & 2003). Here, we mostly focus on Cui & Widom (2003) that describes the authors' method in more detail. This approach treats each transformation as a procedure that can be applied to one or more datasets and produces one or more datasets as output. Cui and Widom present a set of properties that a transformation can have, specifically (a) stability, if it never produces datasets as output without taking any datasets as input, (b) determinism and (c) completeness, if each input data item always contributes to some output data item. The authors assume that all transformations employed in their work are stable and deterministic and proceed to define three major transformation classes of interest: dispatchers, aggregators and black-boxes.

- A transformation is a *dispatcher*, if each input data item produces zero or more output items (with each output item possibly being derived by more than one input data items). A special category of dispatchers are *filters*. A dispatcher is a *filter* if each input item produces either itself or nothing.
- A transformation is an *aggregator,* if it is complete and there exists a unique disjoint partition of the input data set that contributes to some output data item. An aggregator is *context-free* if the output group to which an input data item is mapped can be found by observing the value of this data item alone, independently of the rest of the data items of the input. An aggregator is *key preserving* if all the input originators of an output tuple have the same key value.
- A transformation is a *black-box,* if it is neither a dispatcher nor an aggregator.

Several other sub-categories concerning the mapping of input to output values on the basis of the keys of the input and the output are also defined. The main idea is that if a certain value of an output (input) tuple can directly lead to its respective input (output) tuple(s), then this can be exploited during lineage determination. For example, *backward key transformations* have the property that given an output tuple, one can determine the key of the input tuples that produced it; in this case, lineage is straightforward. In the general case, the lineage determination for dispatchers requires one pass of the input; the lineage determination of aggregators requires several full scans of the input, whereas black boxes have the entire input as their lineage. Given a sequence of transformations in an ETL scenario, it is necessary to keep a set of intermediate results between transformations, in order to be able to determine the data lineage of the output. Appropriate indexes may be used to relate each output to its input. To avoid storing all the intermediate results, the authors propose a *normalization* process that allows the reordering of the transformations in such a way that adjacent transformations share similar properties. Assuming two adjacent transformations *a* and *b*, the idea is that their grouping is beneficial if one can determine the lineage of a tuple in the output of *b* at the input of *a*, without storing any intermediate results. To this end, Cui and Widom propose a greedy algorithm, called *Normalize*, which repeatedly discovers beneficial combinations of adjacent transformations and combines the best pair of transformations.

Theoretical Foundations for ETL Processes

The theoretical underpinnings concerning the internal complexity of simple ETL transformations are investigated by research that concerns the *data exchange* problem. Assume a source and a target schema, a set of mappings that

specify the relationship between the source and the target schema and a set of constraints at the target schema. The problem of data exchange studies whether we can materialize an instance at the target, given an instance at the source, such that all mappings and constraints are respected. Data exchange is a specific variant of the general data integration meta-problem, that requires the materialization of the result at the target, with the ultimate purpose of being able to subsequently pose queries to it, without the luxury of referring back to the source. Due to this inherent characteristic of the problem, we believe that the data exchange is the closest theoretical problem to ETL that we know of.

As Fagin, Kolaitis & Popa (2005) mention, several problems arise around the data exchange problem, with particular emphasis on (a) materializing the solution that reflects the source data as accurately as possible and (b) doing so in an efficient manner. A first important result comes in the identification of the best possible solutions to the problem, called *universal solutions* (Fagin, Kolaitis, Miller, & Popa, 2005). Universal solutions have the good property of having exactly the data needed for the data exchange and can be computed in polynomial time via the chase. It is interesting that if a solution exists, then a universal solution exists, too, and every other solution has a homomorphism to it. At the same time, Fagin et al (TODS 2005) deal with the problem of more than one universal solution for a problem and introduce the notion of a *core*, which is the universal solution of the smallest size. For practical cases of data exchanges, polynomial time algorithms can be used for the identification of the core. It is worth noting that the schema mappings explored are investigated for the case of tuple-generating dependencies, which assure that for each source tuple x, whenever a conjunctive formula applies to it, then there exists a target tuple y, such that a conjunctive formula over x and y applies too (in other words, we can assure that mappings and constraints are respected). To our point of view, this is a starting point for the investigation of more complex mappings that arise in ETL settings.

Further results have to do with the ability to pose queries and obtain certain answers from a data exchange setting as well as with the management of inverse mappings relationships (Fagin, 2007).

Data Cleaning

The area of data cleaning, although inherently related to the ETL process, practically constitutes a different field on its own. Covering this field adequately is well beyond the scope of this paper. The topic that has been in the center of attention of the research community in the area of data cleaning concerns record matching, with a particular emphasis on textual attributes. Record matching refers to the problem of identifying records that represent the *same* object / fact in the real world, with *different* values. Essentially, it is the case of textual fields that presents the major research challenge, since their unstructured nature allows users to perform data entry in arbitrary ways. Moreover, value discrepancies due to problems in the data entry or data processing, as well as different snapshots of the representation of the real world fact in the database contribute to making the problem harder. Typically, the identification of duplicates requires an efficient algorithm for deciding which tuples are to be compared and a distance (or, similarity) metric based on the values of the compared tuples (and possibly, some knowledge by an expert).

For recent, excellent surveys of the field, the reader is encouraged to first refer to a survey by Elmagarmid, Ipeirotis & Verykios (2007) as well as to a couple of tutorials by Koudas & Srivastava (2005) and Koudas, Sarawagi & Srivastava (2006) in the recent past.

Research Efforts Concerning Data Loading Tasks

Typically, warehouse loading tasks take place in a periodic fashion, within a particular time window during which, the system is dedicated to this purpose. Bulk loading is performed (a) during the very first construction of the ware-

house and (b) in an incremental way, during its everyday maintenance. During the latter task, a set of new insertions, deletions and updates arrive at the warehouse, after being identified at the extraction phase and, subsequently transformed and cleaned. This set of data is typically called the 'delta' relation and has to be loaded to the appropriate destination table. In all occasions, loading is performed via vendor-specific bulk loaders that provide maximum performance (see Vassiliadis & Simitsis, 2009 for a discussion on the topic). Also, apart from the dimension tables and the fact tables, indexes and materialized views must also be maintained.

It is interesting that only the loading of views and indexes has been investigated by the research community; in this subsection, we give an overview of important references in the bibliography.

Maintenance of Indexes

Concerning the general setting of index maintenance, Fenk, Kawakami, Markl, Bayer & Osaki (2000) highlight the critical parameters of index bulk loading that include the minimization of random disk accesses, disk I/O, CPU load, and the optimization of data clustering and page filling. As typically happens with spatial indexes, Fenk et al., identify 3 stages in the bulk loading of a multidimensional warehouse index: (a) key calculation for each tuple of the data set, (b) sorting of the tuples on the basis of their keys and (c) loading of the sorted data into the index.

Roussopoulos, Kotidis & Roussopoulos (1997) discuss the efficient construction and bulk load of cube trees. Cube trees and their variants are based on the fundamental idea of finding an efficient data structure for all the possible aggregations of these data. Assume a relation with M attributes, out of which N, $N<M$, can act as grouping attributes for aggregate queries. The idea behind cube trees is the mapping of all the data of all these possible aggregations to a single index that can efficiently answer aggregate range queries. The packing of the empty space is crucial for the compression and efficient query answering of the cube tree. Roussopoulos et al (1997) discuss the bulk loading and maintenance of cube trees that are implemented over packed R-trees. The authors make the sharp observation that individual updates are practically impossible in terms of performance and thus, bulk updates are necessary. The approach is based on sorting the delta increment that is to be loaded and merging it with the existing cube tree. Since all possible aggregates are kept by cube trees, for every delta tuple, all its possible projections are computed and stored in the appropriate buffer. When a buffer is full, it is sorted and then it is staged for the subsequent merge that takes place once all the delta increment has been processed. The authors also highlight that as time passes, the points of all the possible cubes of the multidimensional space are covered with some values, which makes the incremental update even easier. Moreover, since the merging involves three parts (i) the old cube tree, (ii) the delta and (iii) the new cube tree, their merging can be efficiently obtained by using three disks, one for each part. Roussopoulos, Kotidis & Sismanis (1999) discuss this possibility.

Fenk et al. (2000) propose two bulk loading algorithms for the UB-Tree, one for the initial and one for the incremental bulk loading of a UB-tree. The UB-Tree is a multidimensional index, which is used along with specialized query algorithms for the sequential access to the stored data. The UB-Tree clusters data according to a space filling Z–curve. Each point of the multidimensional space is characterized by its Z-address and Z-addresses are organized in disjoined Z-regions mapped to the appropriate disk pages. This way, a tree can be formed and the location of specific points at the disk can be computed. Concerning the abovementioned generic method for bulk loading that Fenk et al have highlighted, in the case of UB-trees, the key is computed by taking the primary key of a tuple and calculating its Z-value and the sorting is performed with external merge sorting. To achieve high page filling, the construction of the UB-tree uses the idea of organizing data in buffers twice as large as disk pages; these buffers

are split in two and one of the two halves is stored as a UB-tree page. The incremental variant of the algorithm tries to identify the correct page for inserting a tuple under consideration.

Dwarfs (Sismanis, Deligiannakis, Roussopoulos & Kotidis, 2002) are indexes built by exploiting an appropriate ordering of the dimensions and the exploitation of common prefixes and suffixes in the data. A dwarf resembles a trie, with one level for each dimension of a cube and appropriate pointers among internal levels. High cardinality dimensions are placed highly in the Dwarf, to quickly reduce the branching factor. Dwarf construction exploits a single, appropriate sorting of data and proceeds in a top-down fashion. Identical sub-dwarfs (i.e., dwarfs generated from the same set of tuples of the fact table) are pinpointed and coalesced. Dwarfs constitute the state-of-the-art, both in querying and in updating sets of aggregate views and, when size and update time are considered, they significantly outperform other approaches.

Materialized View Maintenance

View maintenance is a vast area by itself; covering the topic to the full of its extent is far beyond the focus of this paper. The main idea around view maintenance is developed around the following setting: Assume a set of base tables upon which a materialized view is defined via a query. Changes (i.e., insertions, deletions and updates) occur at the base tables resulting in the need to refresh the contents of the materialized view (a) correctly and (b) as efficiently as possible. The parameters of the problem vary and include:

a. The query class: The area started by dealing with Select-Project-Join views, but the need for aggregate views, views with nested queries in their definition, outerjoins and other more complicated components has significantly extended the field. The query class (along with several assumptions) can also determine whether the materialized view can be updated solely with the changes and its current state, without accessing the base tables.
b. The nature of the changes: apart from simple insertions and deletions, it is an issue whether updates are treated per se or as a pair (delete old value, insert new value). Also, sets of updates as opposed to individual updates can be considered.
c. The number of materialized views that are concurrently been updated: in the context of data warehousing, and ETL in particular, simply dealing with one view being updated is too simplistic. Typically, several materialized views (possibly related in some hierarchical fashion, where one view can be derived from the other) have to be simultaneously refreshed.
d. The way the update is physically implemented: Typically, there are three ways to refresh views, (a) on-update, i.e., the instant a change occurs at the sources, (b) on-demand, i.e., in a deferred way, only when someone poses a query to the view and (c), periodically, which is the typical case for data warehousing, so far.

There are several surveys that give pointers for further reading. The earliest of them was authored by Gupta & Mumick (1995). One can also refer to a note by Roussopoulos (1998) and a chapter by Kotidis (2002). Gupta & Mumick (2006) is a recent paper that discusses maintenance issues for a complicated class of views. The reader is also referred to papers by Mumick, Quass & Mumick (1997), Colby, Kawaguchi, Lieuwen, Mumick & Ross (1997), Labio, Yerneni & Garcia-Molina (1999), and Stanoi, Agrawal & El Abbadi (1999), for the update of groups of views in the presence of updates.

HOLISTIC APPROACHES

In the previous section, we have dealt with operators facilitating tasks that are located in isolation in one of the three main areas of the ETL process. In this section, we take one step back

and give a holistic view to research problems that pertain to the entirety of the ETL process. First, we discuss the problems of optimization and resumption of entire ETL workflows. Second, we visit the practical problem of the lack of a reference benchmark for ETL processes. Finally, we cover the newest topic of research in the area, concerning the near real time refreshment of a data warehouse.

Optimization

The minimization of the execution time of an ETL process is of particular importance, since ETL processes have to complete their task within specific time windows. Moreover, in the unfortunate case where a failure occurs during the execution of an ETL process, there must be enough time left for the resumption of the workflow. Traditional optimization methods are not necessarily applicable to ETL scenarios. As mentioned by Tziovara, Vassiliadis & Simitsis (2007) "*ETL workflows are NOT big queries*: their structure is not a left-deep or bushy tree, black box functions are employed, there is a considerable amount of savepoints to aid faster resumption in cases of failures, and different servers and environments are possibly involved. Moreover, frequently, the objective is to meet specific time constraints with respect to both regular operation and recovery (rather than the best possible throughput)." For all these reasons, the optimization of the execution of an ETL process poses an important research problem with straightforward practical implications.

Simitsis, Vassiliadis & Sellis (ICDE 2005, TKDE 2005) handle the problem of ETL optimization as a state-space problem. Given an original ETL scenario provided by the warehouse designer, the goal of the paper is to find a scenario which is equivalent to the original and has the best execution time possible. Each state is a directed acyclic graph with relations and activities as the nodes and data provider relationships as edges. The authors propose a method that produces states that are equivalent to the original one (i.e., given the same input, they produce the same output) via transitions from one state to another. A transition involves a restructuring of the graph mainly in one of the following ways:

- *swapping* of two consecutive activities if this is feasible, with the goal of bringing highly selective activities towards the beginning of the process (in a manner very similar to the respective query optimization heuristic in relational DBMS's),
- *factorization* of common (or, in the paper's terminology, homologous) activities in a workflow that appear in different paths that end in a binary transformation, with the goal of applying a transformation only once, later in the workflow, possibly to fewer or sorted data,
- *distribution* of common activities (the inverse of factorization) by pushing a transformation that appears late in the workflow towards its start, with the hope that a highly selective activity found after a binary transformation is pushed early enough in the workflow.

Two other transitions, *merge* and *split* are used for special cases. The important problem behind the proposed transitions is that a restructuring of the graph is not always possible. For example, it is important to block the swapping of activities where the operation of the second requires an attribute computed in the first. At the same time, the detection of homologous activities requires the identification of activities with common functionality over data with similar semantics. An ontological mapping of attributes to a common conceptual space facilitates this detection.

The paper uses a very simple cost model to assess the execution cost of a state and presents three algorithms to detect the best possible algorithm. Apart from the exhaustive algorithm that explores the full search space, heuristics are also employed to reduce the search space and speed up the process.

The work of Simitsis et al (ICDE 2005, TKDE 2005) for the optimization of an ETL process at the logical level was complemented

with a paper on the optimization of ETL workflows at the physical level by Tziovara et al (2007). In this paper, the authors propose a method that produces an optimal physical level scenario given its logical representation as an input. Again, the input ETL workflow is considered to be a directed acyclic graph constructed as mentioned above. A logical-level activity corresponds to a *template* i.e., an abstract operation that is customized with schema and parameter information for the ETL scenario under consideration. Each logical-level template is physically implemented by a variety of implementations (much like a relational join is implemented by nested loops, merge-sort, or hash join physical-level operators). Each physical operator is sensitive to the order of the incoming stream of tuples and has a different cost and needs of system resources (e.g., memory, disk space, etc). Again, the problem is modeled as a state-space problem with states representing full implementations of all the logical level activities with their respective physical-level operators. Transitions are of two kinds: (a) replacement of a physical implementation and (b) introduction of *sorter activities* which apply on stored recordsets and sort their tuples according to the values of some, critical for the sorting, attributes. The main idea behind the introduction of sorters is that order-aware implementations can be much faster than their order-neutral equivalents and possibly outweigh the cost imposed by the sorting of data. As with classical query optimization, existing ordering in the storage of incoming data or the reuse of interesting orders in more than one operator can prove beneficial. Finally, this is the first paper where *butterflies* have been used as an experimental tool for the assessment of the proposed method (see the coming section on benchmarking for more details).

The Resumption Problem

An ETL process typically processes several MB's or GB's of data. Due to the complex nature of the process and the sheer amount of data, failures create a significant problem for warehouse administrators – mainly due to the time limits (typically referred to as the *time window*) within which the loading process must be completed. Therefore, the efficient resumption of the ETL process in the case of failures is very important.

Labio, Wiener, Garcia-Molina & Gorelik (2000) are concerned with the issue of efficiently resuming an interrupted workflow. Instead of redoing the workflow all over from the beginning, the authors propose a resumption algorithm, called *DR*, based on the fundamental observation that whenever an activity outputs data in an ordered fashion, then its resumption can start right where it was interrupted. Activities are practically treated as black boxes (where only the input and output are of interest) and a tree of activities is used to model the workflow. Each path of the tree is characterized on the relationship of output to input tuples and on the possibility of ignoring some tuples. Each transformation in an ETL process is characterized with respect to a set of properties that concern (a) the extent to which an input tuple produces more than one output tuples (if not, this can be exploited at resumption time), (b) the extent to which a prefix or a suffix of a transformation can be produced by a prefix or a suffix of the input (in which case, resumption can start from the last tuple under process at the time of failure), (c) the order produced by the transformation (independently of the input's order) and the deterministic nature of the transformation. Combinations of these properties are also considered by the authors. Moreover, these properties are not defined only for transformations in isolation, but also, they are generalized for sequences of transformations, i.e., the whole ETL process.

The resumption algorithm has two phases: (a) *design*, where the activities of the workflow are characterized with respect to the aforementioned properties and (b) *resumption*, which is based on the previous characterization and invoked in the event of failure.

- Design constructs a workflow customized to execute the resumption of the original

workflow. To this end, re-extraction procedures are assigned to extractors; these procedures regulate whether all or part of the input will be extracted from the sources. These re-extraction procedures are complemented with filters that are responsible for blocking source or intermediate tuples that have already been processed – and most importantly, stored – during the regular operation of the ETL process.

- Resume consists of the assignment of the correct values that must be assigned to the resumption workflow's filters, so that the tuples that are already stored in the warehouse are blocked. Then, Resume performs the application of the re-extraction procedures. Subsequently, the load of the warehouse continues as in regular operation.

Benchmarking

Unfortunately, the area of ETL processes suffers from the absence of a thorough benchmark that puts tools and algorithms to the test, taking into consideration parameters like the complexity of the process, the data volume, the amount of necessary cleaning and the computational cost of individual activities of the ETL workflows.

The Transaction Processing Council has proposed two benchmarks for the area of decision support. The TPC-H benchmark (TPC-H, 2008) is a decision support benchmark that consists of a suite of business-oriented ad-hoc queries and concurrent data modifications. The database describes a sales system, keeping information for the parts and the suppliers, and data about orders and the supplier's customers. The relational schema of TPC-H is not a typical warehouse star or snowflake schema; on the contrary, apart from a set of tables that are clearly classified as dimension tables, the facts are organized in a combination of tables acting as bridges and fact tables. Concerning the ETL process, the TPC-H requires the existence of very simple insertion and deletion SQL statements that directly modify the contents of the LINEITEM and ORDERS warehouse tables. Clearly, TPC-H is not related to ETL, since there is no workflow of cleanings or transformations, no value computations and no routing of data from the sources to the appropriate targets in the warehouse.

TPC-DS is a new Decision Support (DS) workload being developed by the TPC (TPC-DS, 2005). This benchmark models the decision support system of a retail product supplier, including queries and data maintenance. The relational schema of this benchmark is more complex than the schema presented in TPC-H. TPC-DS involves six star schemata (with a large overlap of shared dimensions) standing for Sales and Returns of items purchased via three sales channels: a Store, a Catalog and the Web channel. The structure of the schemata is more natural for data warehousing than TPC-H; still, the schemata are neither pure stars, nor pure snowflakes. The dimensions follow a snowflake pattern, with a different table for each level; nevertheless, the fact table has foreign keys to all the dimension tables of interest (resulting in fast joins with the appropriate dimension level whenever necessary). TPC-DS provides a significantly more sophisticated palette of refreshment operations for the data warehouse than TPC-H. There is a variety of maintenance processes that insert or delete facts, maintain inventories and refresh dimension records, either in a history keeping or in a non-history keeping method. To capture the semantics of the refreshment functions, warehouse tables are pseudo-defined as views over the sources. The refreshment scenarios of TPC-DS require the usage of functions for transformations and computations (with date transformations and surrogate key assignments being very popular). Fact tables are also populated via a large number of inner and outer joins to dimension tables. Overall, TPC-DS is a significant improvement over TPC-H in terms of benchmarking the ETL process; nevertheless, it still lacks the notion of large workflows of activities with schema and value transformations, row routing and other typical ETL features.

A first academic effort for the benchmarking of warehouses is found in Darmont, Ben-

tayeb, & Boussaïd (2005). Still, the benchmark explicitly mentions ETL processes as future work and does not address the abovementioned problems. A second approach towards a benchmark for ETL processes is presented in Vassiliadis, Karagiannis, Tziovara, Simitsis (2007). The authors provide a rough classification of template structures, which are called *butterflies*, due to their shape. In the general case, a butterfly comprises three elements. First, a butterfly comprises a *left wing* where source data are successively combined (typically via join operations) towards a central point of storage (called the *body* of the butterfly). Finally, the *right wing* comprises the routing and aggregation of the tuples that arrived at the body towards materialized views (also simulating reports, spreadsheets etc.). Depending on the shape of the workflow (where one of the wings might be missing), the authors propose several template workflow structures and a concrete set of scenarios that are materializations of these templates.

Near-Real-Time ETL

Traditionally, the data warehouse refreshment process has been executed in an off-line mode, during a nightly time window. Nowadays, business needs for on-line monitoring of source side business processes drive the request for 100% data freshness at the user's reports at all times. Absolute freshness (or real time warehousing as abusively mentioned) practically contradicts with one of the fundamental reasons for separating the OLTP systems from reporting tasks for reasons of load, lock contention and efficiency (at least when large scale is concerned). Still, due to the user requests, data warehouses cannot escape their transformation to data providers for the end users with an increasing rate of incoming, fresh data. For all these reasons, a compromise is necessary and as a result, the refreshment process moves to periodic refresh operations with a period of hours or even minutes (instead of days). This brings *near real time data warehousing* in the stage. In this subsection, we will review some research efforts towards this direction.

Karakasidis, Vassiliadis & Pitoura (2005), propose a framework for the implementation of near real time warehouse (called "active" data warehousing by the authors). Several goals are taken into consideration by the authors, and specifically: (a) minimal changes in the software configuration of the source, (b) minimal overhead on the source due to the continuity of data propagation, (c) the possibility of smoothly regulating the overall configuration of the environment in a principled way. The architecture of the system is based on pipelining: each activity behaves as a queue (and thus, it called an ETL queue) that periodically checks the contents of its queue buffers and passes a block of tuples to a subsequent queue once they are appropriately processed (i.e., filtered or transformed). The queues of an ETL workflow form a queue network and pipelining takes place. The paper explores a few other possibilities: (a) queue theory is used for the prediction of the behavior of the queue network, (b) a legacy application over ISAM files was modified with minimal software changes and (c) web services were employed at the warehouse end to accept the final blocks of tuples and load them to the warehouse. The latter performed reasonably well, although the lack of lightweightness in web service architectures poses an interesting research challenge.

The work by Luo, Naughton, Ellmann & Watzke (2006) deals with the problem of continuous maintenance of materialized views. The continuous loading of the warehouse with new or updated source data is typically performed via concurrent sessions. In this case, the existence of materialized views that are defined as joins of the source tables may cause deadlocks (practically, the term 'source tables' should be understood as referring to cleansed, integrated "replicas" of the sources within the warehouse). This is due to the fact that the maintenance of the view due to an update of source R_1 may require a lookup to source relation R_2 for the match of new delta. If R_2 is updated concurrently with R_1, then a deadlock may occur. To avoid

deadlock, Luo et al. propose the reordering of transactions in the warehouse. The authors assume that the refreshment process operates over join or aggregate join materialized views. The authors suggest several rules for avoiding deadlocks. Specifically, the authors require that (a) at any time, only one of the relations of a join view is updated, (b) data coming from the sources are not randomly assigned to update sessions, but rather, all the modifications of a tuple are routed to the same refreshment session, and, (c) the traditional 2PL, tuple-level locking mechanism is replaced by some higher level protocol. Individual modifications to the same relation are grouped in transactions and a scheduling protocol for these transactions is proposed, along with some starvation avoidance heuristics. A second improvement that Luo et al. propose is the usage of "pre-aggregation" for aggregate materialized views, which practically involves the grouping of all the modifications of a certain view tuple in the same transaction to one modification with their net effect. This can easily be achieved by sorting the individual updates over their target tuple in the aggregate materialized view. The experimental assessment indicates that the throughput is significantly increased as an effect of the reordering of transactions – especially as the number of modifications per transaction increases.

Thiele, Fischer & Lehner (2007) discuss the problem of managing the workload of the warehouse in a near real time warehouse (called real time warehouse by the authors). The authors discuss a load-balancing mechanism that schedules the execution of query and update transactions according to the preferences of the users. There are two fundamental, conflicting goals that the scheduling tries to reconcile. On the one hand, users want efficient processing of their queries and low response time. This goal is referred to as Quality of Service by the authors. On the other hand, the users also want the warehouse data to be as up-to-date as possible with respect to their originating records at the sources. This goal is referred to as Quality of Data by the authors. Two queues are maintained by the algorithm, one for the queries and one for the update transactions. Since a reconciliation must be made between two conflicting requirements, the authors assume that queries are tagged with two scores, one for the requirement for freshness and another for the requirement of query efficiency. To enable the near real time loading of the warehouse, Thiele et al. propose a two-level scheduling mechanism. The first level of scheduling is dedicated in deciding whether a user query or an update transaction will be executed; this decision is based on the sum of the scores of both requirements for all the queries. The winner sum determines if an update or a query will be executed. The second level of scheduling resolves which transaction will be executed. To avoid implications with data correctness and to serve both goals better, the authors resolve in two scheduling guidelines. If an update transaction is to be executed, then, it should be the one related to the data that are going to be queried by a query at the beginning of the query queue. On the other hand, if a query is to be executed (i.e., efficiency has won the first level of the scheduling contest), then the query with a higher need for Quality of Service is picked. To avoid starvation of queries with low preference to quality of service, the QoS tags of all queries that remain in the queue are increased after each execution of a query.

Thomsen, Pedersen & Lehner (2008) discuss a loader for near real time, or *right time* data warehouses. The loader tries to synchronize the loading of the warehouse with queries that require source data with a specific freshness guarantee. The idea of loading the data *when* they are needed (as opposed to *before* they are needed) produces the notion of right time warehousing. The architecture of the middleware discussed by the authors involves three tiers. The first tier concerns the data producer, at the source side. The middleware module provided for the producer captures the modification operations "insert" and "commit" for JDBC statements. The user at the source side can decide whether committed data are to become available for the warehouse (in this case, this is referred to as materialization by the authors). On insert, the new source values are cached in

a source buffer and wait to be propagated to the next tier, which is called catalyst. Apart from that, the source middleware provides callbacks for regulating its policy as steal or no steal with respect to the flushing of committed data to the hard disk at the source side, as well as a third option for regulating its policy on the basis of its current load or elapsed idle time. The catalyst side, at the same time, also buffers data for the consumer. An interesting property of the catalyst is the guarantee of freshness for the user. So, whenever a warehouse user requests data, he can set a time interval for the data he must have; then the catalyst check its own buffers and communicates with the source-side middleware to compile the necessary data. A precise synchronization of the data that must be shed from main memory via appropriate main memory indexing is also discussed by the authors. The third tier, at the warehouse side, operates in a REPEATABLE READ isolation mode and appropriately locks records (via a shared lock) so that the catalyst does not shed them during their loading to the warehouse. Finally, the authors discuss a simple programmatic facility via appropriate view definitions to allow the warehouse user retrieve the freshest data possible in a transparent way.

Polyzotis, Skiadopoulos, Vassiliadis, Simitsis & Frantzell (2007, 2008) deal with an individual operator of the near real time ETL process, namely the join of a continuous stream of updates generated at the sources with a large, disk resident relation at the warehouse side, under the assumption of limited memory. Such a join can be used in several occasions, such as surrogate key assignment, duplicate detection, simple data transformations etc. To this end, a specialized join algorithm, called MeshJoin is introduced. The main idea is that the relation is continuously brought to main memory in scans of sequential blocks that are joined with the buffered stream tuples. A precise expiration mechanism for the stream tuples guarantees the correctness of the result. The authors propose an analytic cost model that relates the stream rate and the memory budget. This way, the administrator can tune the operation of the algorithm, either in order to maximize throughput for a given memory budget, or in order to minimize the necessary memory for a given stream rate. In the case of thrashing, an approximate version of the algorithm with load shedding strategies that minimize the loss of output tuples is discussed. MeshJoin makes no assumption on the order, indexing, join condition and join relationship of the joined stream and relation; at the same time it relates the stream rate with the available memory budget and gives correctness guarantees for an exact result if this is possible, or, allows a lightweight approximate result otherwise.

SYSTEMS

Industrial systems for ETL are provided both by the major database vendors and the individual ETL-targeted vendors. Popular tools include Oracle Warehouse Builder (2008), IBM Datastage (2008), Microsoft Integration Services (2008) and Informatica PowerCenter (2008). There is an excellent survey by Barateiro & Galhardas (2005) that makes a thorough discussion and feature comparison for ETL tools both of academic and industrial origin. Friedman, Beyer & Bitterer (2007) as well as Friedman & Bitterer (2007) give two interesting surveys of the area from a marketing perspective. In this section, we will discuss only academic efforts related to ETL systems.

AJAX

The *AJAX* system (Galhardas, Florescu, Shasha & Simon, 2000) is a data cleaning tool developed at INRIA France that deals with typical data quality problems, such as duplicate identification, errors due to mistyping and data inconsistencies between matching records. This tool can be used either for a single source or for integrating multiple data sources. AJAX provides a framework wherein the logic of a data cleaning program is modeled as a directed graph of data transformations that start from some input source data. Four types of data transformations are supported:

- *Mapping transformations* standardize data formats (e.g., date format) or simply merge or split columns in order to produce more suitable formats.
- *Matching transformations* find pairs of records that most probably refer to same object. These pairs are called *matching pairs* and each such pair is assigned a similarity value.
- *Clustering transformations* group together matching pairs with a high similarity value by applying a given grouping criteria (e.g., by transitive closure).
- *Merging transformations* are applied to each individual cluster in order to eliminate duplicates or produce new records for the resulting integrated data source.

AJAX also provides a declarative language for specifying data cleaning programs, which consists of SQL statements enriched with a set of specific primitives to express mapping, matching, clustering and merging transformations. Finally, a interactive environment is supplied to the user in order to resolve errors and inconsistencies that cannot be automatically handled and support a stepwise refinement design of data cleaning programs. The linguistic aspects and the theoretic foundations of this tool can be found in Galhardas, Florescu, Shasha, Simon & Saita (2001), and Galhardas, Florescu, Shasha & Simon (1999), where apart from the presentation of a general framework for the data cleaning process, specific optimization techniques tailored for data cleaning applications are discussed.

ARKTOS

ARKTOS (Vassiliadis et al., Information Systems 2005) is an ETL tool that has prototypically been implemented with the goal of facilitating the design, the (re-)use, and the optimization of ETL workflows. Arktos is based on the metamodel of Vassiliadis et al., (Information Systems 2005) for representing ETL activities and ETL workflows and its two key features are (a) the extensibility mechanisms for reusing transformations and (b) the close linkage to formal semantics and their representation. ARKTOS is accompanied by an ETL library that contains template code of built-in functions and maintains template code of user-defined functions. Template activities are registered in the system and they can be reused for the specification of a scenario (either graphically, or via forms and declarative languages). The customization process results in producing an ETL scenario which is a DAG of ETL activities, each specified as a parameterized software module, having instantiated input and output schemata, concrete parameters, and a special-purpose schema for problematic records. The set of templates is extensible to allow users register their own frequently used transformations.

ARKTOS also offers zoom-in/zoom-out capabilities. The designer can deal with a scenario in two levels of granularity: (a) at the *entity* or *zoom-out level*, where only the participating recordsets and activities are visible and their provider relationships are abstracted as edges between the respective entities, or (b) at the *attribute* or *zoom-in level*, where the user can see and manipulate the constituent parts of an activity, along with their respective providers at the attribute level.

Finally, it is noteworthy that the model of Vassiliadis et al., (Information Systems 2005) comes with a mechanism for expressing the semantics of activities in LDL. The expression of semantics can be done both at the template and the instance level and a macro language is discussed in the paper for this purpose. The approach is based on the fundamental observation that the LDL description can be mapped to a useful graph representation of the internals of an activity, which allows both the visualization and the measurement of interesting properties of the graph.

HumMer - Fusion

HumMer (Naumann, Bilke, Bleiholder & Weis, 2006) is a tool developed in the Hasso-Plattner Institute in Potsdam that deals with the problem of data fusion. Data fusion is the task of identify-

ing and resolving different representations of the same object of the real world. HumMer splits the process in three steps: (a) schema alignment, (b) duplicate detection and (c) the core of the data fusion process. Schema alignment deals with the problem of schema inconsistencies. Assuming the user is in possession of different data sets representing the same entities of the real world, HumMer is equipped with a dedicated module that detects a small sample of duplicates in these data sets and tries to find schema similarities. Duplicate detection is performed via comparing tuples over a similarity threshold. Once inconsistencies at the schema and tuple level have been resolved, it is time for the resolution of value inconsistencies, which is referred to as data fusion by Naumann et al (2006). FuSem (Bleiholder, Draba & Naumann, 2007) is an extension of HumMer with the purpose of interactively facilitating the data fusion process. Assuming that the user has different data sets representing the same real world entities, FuSem allows the SQL querying of these data sources, their combination through outer join and union operations and the application of different tuple matching operators. The extension of SQL with the FYSE BY operator, proposed by Bleiholder & Naumann (2005) is also part of FuSem. FYSE BY allows both the alignment of different relations in terms of schemata and the identification of tuples representing the same real world object. An important feature of FuSem is the visualization of results, which is primarily based on different representations of Venn diagrams for the involved records and the interactive exploration of areas where two data sets overlap or differ.

Potter's Wheel

Raman & Hellerstein (2000, 2001) present the *Potter's Wheel* system, which is targeted to provide interactive data cleaning to its users. The system offers the possibility of performing several algebraic operations over an underlying data set, including *format* (application of a function), *drop, copy, add* a column, *merge* delimited columns, *split* a column on the basis of a regular expression or a position in a string, *divide* a column on the basis of a predicate (resulting in two columns, the first involving the rows satisfying the condition of the predicate and the second involving the rest), *selection* of rows on the basis of a condition, *folding* columns (where a set of attributes of a record is split into several rows) and *unfolding*. Optimization algorithms are also provided for the CPU usage for certain classes of operators. The general idea behind Potter's Wheel is that users build data transformations in iterative and interactive way. Specifically, users gradually build transformations to clean the data by adding or undoing transformations on a spreadsheet-like interface; the effect of a transformation is shown at once on records visible on screen. These transformations are specified either through simple graphical operations, or by showing the desired effects on example data values. In the background, Potter's Wheel automatically infers structures for data values in terms of user-defined domains, and accordingly checks for constraint violations. Thus, users can gradually build a transformation as discrepancies are found, and clean the data without writing complex programs or enduring long delays.

CONCLUDING REMARKS

This survey has presented the research work in the field of Extraction-Transformation-Loading (ETL) processes and tools. The main research goals around which research has been organized so far can be summarized as follows.

a. The first goal concerns the construction of commonly accepted conceptual and logical modeling tools for ETL processes, with a view to a standardized approach.
b. The second goal concerns the efficiency of individual ETL operators. To structure the discussion better, we have organized the discussion around the three main parts of the E-T-L triplet, and examined problems that fall within each of these stages. So far, research has come up with interesting solu-

tions for the detection of differences in two snapshots of data, specific data transformations, duplicate detection mechanisms and the practical and theoretical investigation of the lineage of warehouse data.

c. A third research goal has been to devise algorithms for the efficient operation of the entire ETL process. Specifically, research has some first results for the optimization and resumption of entire ETL processes as well as some first investigations towards (near) real time ETL.
d. A fourth goal of the academic world has been the construction of tools for the facilitation of the ETL process.

Apparently, the field is quite new, and there is still too much to be done. In the sequel, we present a personal viewpoint on the possible advancements that can be made in the field.

Starting with the traditional ETL setting, there are quite a few research opportunities. Individual operators are still far from being closed as research problems. The extraction of data still remains a hard problem, mostly due to the closed nature of the sources. The loading problem is still quite unexplored with respect to its practical aspects. The optimization and resumption problems -along with the appropriate cost models- are far from maturity. The absence of a benchmark is hindering future research (that lacks a commonly agreed way to perform experiments). The introduction of design patterns for warehouse schemata that take ETL into consideration is also open research ground.

At the same time, the presence of new heavily parallelized processors and the near certainty of disrupting effects in hardware and disk technology in the immediate future put all the database issues of efficiency back on the table. ETL cannot escape the rule and both individual transformations as well as the whole process can be reconsidered in the presence of these improvements. Most importantly, all these research opportunities should be viewed via the looking glass of near real time ETL, with the need for completeness and freshness of data to be pressing from the part of the users.

As an overall conclusion, we believe that design, algorithmic and theoretical results in the field of ETL processes are open to exploration both due to the present problems and on the basis of the changing environment of computer science, databases and user needs.

REFERENCES

Barateiro, J., & Galhardas, H. (2005). A Survey of Data Quality Tools. *Datenbank-Spektrum 14*, 15-21

Bleiholder, J., & Naumann, F. (2005). *Declarative Data Fusion - Syntax, Semantics, and Implementation*. 9th East European Conference on Advances in Databases and Information Systems (ADBIS 2005), pp.: 58-73, Tallinn, Estonia, September 12-15, 2005.

Bleiholder, J., Draba, K., & Naumann, F. (2007). *FuSem - Exploring Different Semantics of Data Fusion*. Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007), pp.: 1350-1353, University of Vienna, Austria, September 23-27, 2007.

Carreira P., Galhardas, H., Pereira, J., Martins, F., & Silva, M. (2007). *On the performance of one-to-many data transformations*. Proceedings of the Fifth International Workshop on Quality in Databases (QDB 2007), pp.: 39-48, in conjunction with the VLDB 2007 conference, Vienna, Austria, September 23, 2007

Carreira, P., Galhardas, H., Lopes A., & Pereira J. (2007). One-to-many data transformations through data mappers. *Data Knowledge Engineering*, *62*, 3, 483-503

Carreira, P., Galhardas, H., Pereira, J., & Lopes, A. (2005). *Data Mapper: An Operator for Expressing One-to-Many Data Transformations*. 7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2005), pp.: 136-145, Copenhagen, Denmark, August 22-26, 2005

Carreira, P., Galhardas, H., Pereira, J., & Wichert, A. (2007). *One-to-many data transformation operations - optimization and execution on an RDBMS*. Proceedings of the Ninth International Conference

on Enterprise Information Systems, Volume DISI, ICEIS (1) 2007, pp.: 21-27, Funchal, Madeira, Portugal, June 12-16, 2007

Colby, L., Kawaguchi, A., Lieuwen, D., Mumick, I., & Ross, K. (1997). *Supporting Multiple View Maintenance Policies*. In Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD 1997), pp.: 405-416, May 13-15, 1997, Tucson, Arizona, USA

Cui, Y., & Widom, J. (2001). *Lineage Tracing for General Data Warehouse Transformations*. Proceedings of 27th International Conference on Very Large Data Bases (VLDB 2001), pp.: 471-480, September 11-14, 2001, Roma, Italy

Cui, Y., & Widom, J. (2003). Lineage tracing for general data warehouse transformations. *VLDB Journal 12,* 1, 41-58

Cunningham, C., Galindo-Legaria, C., & Graefe, G. (2004). *PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS*. Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB 2004), pp. 998-1009, Toronto, Canada, August 31 - September 3 2004.

Darmont, J., Bentayeb, F., & Boussaïd, O. (2005). *DWEB: A Data Warehouse Engineering Benchmark*. Proceedings 7th International Conference Data Warehousing and Knowledge Discovery (DaWaK 2005), pp. 85–94, Copenhagen, Denmark, August 22-26 2005

Davidson, S., & Kosky, A. (1999). Specifying Database Transformations in WOL. *Bulletin of the Technical Committee on Data Engineering*, *22*, 1, 25-30.

Elmagarmid, A., Ipeirotis, P., & Verykios, V. (2007). Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, *19*, 1, 1-16

Fagin, R. (2007). Inverting schema mappings. *ACM Transactions on Database Systems*, *32*, 4, 25:1-25:53

Fagin, R., Kolaitis, P., & Popa, L. (2005). Data exchange: getting to the core. *ACM Transactions on Database Systems*, *30*, 1, 174-210

Fagin, R., Kolaitis, P., Miller, R., & Popa, L. (2005). Data exchange: semantics and query answering. *Theoretical Computer Science*, *336*, 1, 89-124

Fenk, R., Kawakami, A., Markl, V., Bayer, R., & Osaki, S. (2000) Bulk Loading a Data Warehouse Built Upon a UB-Tree. Proceedings 2000 International Database Engineering and Applications Symposium (IDEAS 2000),pp.: 179-187, September 18-20, 2000, Yokohoma, Japan

Friedman, T., & Bitterer, A. (2007). *Magic Quadrant for Data Quality Tools, 2007*. Gartner RAS Core Research Note G00149359, 29 June 2007. Available at http://mediaproducts.gartner.com/reprints/businessobjects/149359.html (last accessed 23 July 2008)

Friedman, T., Beyer, M., & Bitterer, A. (2007). *Magic Quadrant for Data Integration Tools, 2007*. Gartner RAS Core Research Note G00151150, 5 October 2007. Available at http://mediaproducts.gartner.com/reprints/oracle/151150.html (last accessed 23 July 2008)

Galhardas, H., Florescu, D., Shasha, D., & Simon, E. (1999). *An Extensible Framework for Data Cleaning*. Technical Report INRIA 1999 (RR-3742).

Galhardas, H., Florescu, D., Shasha, D., & Simon, E. (2000). *Ajax: An Extensible Data Cleaning Tool*. In Proc. ACM SIGMOD International Conference on the Management of Data, pp. 590, Dallas, Texas.

Galhardas, H., Florescu, D., Shasha, D., Simon, E. & Saita, C. (2001). *Declarative Data Cleaning: Language, Model, and Algorithms*. Proceedings of 27th International Conference on Very Large Data Bases (VLDB 2001), pp.: 371-380, September 11-14, 2001, Roma, Italy

Gupta, A., & Mumick, I. (1995). Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin*, *18*, 2, 3-18

Gupta, H., & Mumick, I. (2006). Incremental maintenance of aggregate and outerjoin expressions. *Information Systems*, *31*, 6, 435-464

IBM. WebSphere DataStage. Product's web page at http://www-306.ibm.com/software/data/integration/datastage/Last accessed 21 July 2008.

Informatica. PowerCenter. Product's web page at http://www.informatica.com/products_services/powercenter/Pages/index.aspxLast accessed 21 July 2008.

Karakasidis, A., Vassiliadis, P., & Pitoura, E. (2005). *ETL Queues for Active Data Warehousing*. In Proc.

2nd International Workshop on Information Quality in Information Systems (IQIS 2005), co-located with ACM SIGMOD/PODS 2005, June 17, 2005, Baltimore, MD, USA.

Kimbal, R., Reeves, L., Ross, M., & Thornthwaite, W. (1998). *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*. John Wiley & Sons.

Kimball, R., & Caserta, J. (2004). *The Data Warehouse ETL Toolkit*. Wiley Publishing, Inc.

Kotidis, Y. (2002). Aggregate View Management in Data Warehouses. In *Handbook of Massive Data Sets* (pages 711-742). *Kluwer Academic Publishers,* ISBN 1-4020-0489-3

Koudas, N., & Srivastava, D. (2005). Approximate Joins: Concepts and Techniques. Tutorial at the 31st International Conference on Very Large Data Bases (VLDB 2005), pp.: 1363, Trondheim, Norway, August 30 - September 2, 2005. Slides available at http://queens.db.toronto.edu/~koudas/docs/AJ.pdf (last accessed 23 July 2008)

Koudas, N., Sarawagi, S., & Srivastava, D. (2006). Record linkage: similarity measures and algorithms. Tutorial at the ACM SIGMOD International Conference on Management of Data (SIGMOD 2006), pp.:802-803, Chicago, Illinois, USA, June 27-29, 2006. Slides available at http://queens.db.toronto.edu/~koudas/docs/aj.pdf (last accessed 23 July 2008)

Labio, W., & Garcia-Molina, H. (1996). *Efficient Snapshot Differential Algorithms for Data Warehousing*. In Proceedings of 22nd International Conference on Very Large Data Bases (VLDB 1996), pp. 63-74, September 3-6, 1996, Mumbai (Bombay), India

Labio, W., Wiener, J., Garcia-Molina, H., & Gorelik, V. (2000). *Efficient Resumption of Interrupted Warehouse Loads*. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 2000), pp. 46-57, Dallas, Texas, USA

Labio, W., Yerneni, R., & Garcia-Molina, H. (1999). *Shrinking the Warehouse Update Window*. Proceedings ACM SIGMOD International Conference on Management of Data, (SIGMOD 1999), pp.: 383-394, June 1-3, 1999, Philadelphia, Pennsylvania, USA

Lindsay, B., Haas, L., Mohan, C., Pirahesh, H., & Wilms, P. (1986). *A Snapshot Differential Refresh Algorithm*. Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data (SIGMOD 1998), pp: 53-60, Washington, D.C., May 28-30, 1986.

Luján-Mora, S., Vassiliadis, P., & Trujillo, J. (2004). *Data Mapping Diagrams for Data Warehouse Design with UML*. In Proc. 23rd International Conference on Conceptual Modeling (ER 2004), pp. 191-204, Shanghai, China, 8-12 November 2004.

Luo, G., Naughton, J., Ellmann, C., & Watzke, M. (2006). *Transaction Reordering and Grouping for Continuous Data Loading*. First International Workshop on Business Intelligence for the Real-Time Enterprises (BIRTE 2006), pp. 34-49. Seoul, Korea, September 11, 2006

Microsoft. SQL Server Integration Services. Product's web page at http://www.microsoft.com/sql/technologies/integration/default.mspxLast accessed 21 July 2008.

Mumick, I., Quass, D., & Mumick, B. (1997) *Maintenance of Data Cubes and Summary Tables in a Warehouse*. In Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD 1997), pp.: 100-111, May 13-15, 1997, Tucson, Arizona, USA

Naumann, F., Bilke, A., Bleiholder, J., & Weis, M. (2006). Data Fusion in Three Steps: Resolving Schema, Tuple, and Value Inconsistencies. *IEEE Data Engineering Bulletin*, *29,* 2, 21-31

Oracle. Oracle Warehouse Builder. Product's web page at http://www.oracle.com/technology/products/warehouse/index.html Last accessed 21 July 2008.

Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., & Frantzell, N. (2007). *Supporting Streaming Updates in an Active Data Warehouse*. In Proc. 23rd International Conference on Data Engineering (ICDE 2007), pp 476-485, Constantinople, Turkey, April 16-20, 2007.

Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., & Frantzell, N. (2008). Meshing Streaming Updates with Persistent Data in an Active Data Warehouse. *IEEE Transactions on Knowledge and Data Engineering*, *20*, 7, 976-991

Raman, V., & Hellerstein, J. (2000). *Potters Wheel: An Interactive Framework for Data Cleaning and Transformation*. Technical Report University of California at Berkeley, Computer Science Divi-

sion, 2000. Available at http://www.cs.berkeley.edu/~rshankar/papers/pwheel.pdf

Raman, V., & Hellerstein, J. (2001). *Potter's Wheel: An Interactive Data Cleaning System*. In Proceedings of 27th International Conference on Very Large Data Bases (VLDB 2001), pp. 381-390, Roma, Italy.

Roth M., & Schwarz P. (1997). *Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources*. In Proceedings of 23rd International Conference on Very Large Data Bases (VLDB 1997), pp. 266-275, August 25-29, 1997, Athens, Greece

Roussopoulos, N. (1998). Materialized Views and Data Warehouses. *SIGMOD Record, 27*, 1, 21-26

Roussopoulos, N., Kotidis, Y., & Roussopoulos, M. (1997). Cubetree: Organization of and Bulk Updates on the Data Cube. Proceedings ACM SIGMOD International Conferenceon Management of Data (SIGMOD 1997), pp. 89-99, May 13-15, 1997, Tucson, Arizona, USA

Roussopoulos, N., Kotidis, Y., & Sismanis, Y. (1999). The Active MultiSync Controller of the Cubetree Storage Organization. Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD 1999), pp.: 582-583, June 1-3, 1999, Philadelphia, Pennsylvania, USA

Shu, N., Housel, B., Taylor, R., Ghosh, S., & Lum, V. (1977). EXPRESS: A Data EXtraction, Processing, and REStructuring System. *ACM Transactions on Database Systems, 2*, 2, 134-174.

Simitsis, A. (2005). Mapping conceptual to logical models for ETL processes. In Proceedings of ACM 8th International Workshop on Data Warehousing and OLAP (DOLAP 2005), pp.: 67-76 Bremen, Germany, November 4-5, 2005

Simitsis, A., & Vassiliadis, P. (2008). A Method for the Mapping of Conceptual Designs to Logical Blueprints for ETL Processes. *Decision Support Systems*, *45*, 1, 22-40.

Simitsis, A., Vassiliadis, P., & Sellis, T. (2005). *Optimizing ETL Processes in Data Warehouses*. Proceedings 21st Int. Conference on Data Engineering (ICDE 2005), pp. 564-575, Tokyo, Japan, April 2005.

Simitsis, A., Vassiliadis, P., & Sellis, T. (2005). State-Space Optimization of ETL Workflows. *IEEE Transactions on Knowledge and Data Engineering*, *17*, 10, 1404-1419

Simitsis, A., Vassiliadis, P., Terrovitis, M., & Skiadopoulos, S. (2005). *Graph-Based Modeling of ETL activities with Multi-level Transformations and Updates*. In Proc. 7th International Conference on Data Warehousing and Knowledge Discovery 2005 (DaWaK 2005), pp. 43-52, 22-26 August 2005, Copenhagen, Denmark.

Sismanis, Y., Deligiannakis, A., Roussopoulos, N., & Kotidis, Y. (2002). Dwarf: shrinking the PetaCube. roceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002), 464-475, Madison, Wisconsin, June 3-6, 2002

Skoutas, D., & Simitsis, A., (2006). *Designing ETL processes using semantic web technologies*. In Proceedings ACM 9th International Workshop on Data Warehousing and OLAP (DOLAP 2006), pp.:67-74, Arlington, Virginia, USA, November 10, 2006

Skoutas, D., & Simitsis, A., (2007). Flexible and Customizable NL Representation of Requirements for ETL processes. In Proceedings 12th International Conference on Applications of Natural Language to Information Systems (NLDB 2007), pp.: 433-439, Paris, France, June 27-29, 2007

Skoutas, D., & Simitsis, A., (2007). Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data. *Int. Journal of Semantic Web Information Systems (IJSWIS) 3*, 4, 1-24

Stanoi, I., Agrawal, D., & El Abbadi, A. (1999). *Modeling and Maintaining Multi-View Data Warehouses*. Proceedings 18th International Conference on Conceptual Modeling (ER 1999), pp.: 161-175, Paris, France, November, 15-18, 1999

Stöhr, T., Müller, R., & Rahm, E. (1999). *An integrative and Uniform Model for Metadata Management in Data Warehousing Environments*. In Proc. Intl. Workshop on Design and Management of Data Warehouses (DMDW 1999), pp. 12.1 – 12.16, Heidelberg, Germany, (1999).

Thiele, M., Fischer, U., & Lehner, W. (2007). *Partition-based workload scheduling in living data warehouse environments*. In Proc. ACM 10th International Workshop on Data Warehousing and OLAP (DOLAP 2007), pp. 57-64, Lisbon, Portugal, November 9, 2007.

Thomsen, C., Pedersen, T., & Lehner, W. (2008). *RiTE: Providing On-Demand Data for Right-Time*

Data Warehousing. Proceedings of the 24th International Conference on Data Engineering (ICDE 2008), pp 456 – 465, April 7-12, 2008, Cancun, Mexico.

TPC. The TPC-DS benchmark. Transaction Processing Council. Available at http://www.tpc.org/tpcds/default.asp (last accessed at 24 July 2008)

TPC. The TPC-H benchmark. Transaction Processing Council. Available at http://www.tpc.org/ (last accessed at 24 July 2008)

Trujillo, J., & Luján-Mora, S. (2003). *A UML Based Approach for Modeling ETL Processes in Data Warehouses*. In Proceedings of 22nd International Conference on Conceptual Modeling (ER 2003), pp. 307-320, Chicago, IL, USA, October 13-16, 2003.

Tziovara, V., Vassiliadis, P., & Simitsis, A. (2007). *Deciding the Physical Implementation of ETL Workflows*. Proceedings ACM 10th International Workshop on Data Warehousing and OLAP (DOLAP 2007), pp. 49-56, Lisbon, Portugal, 9 November 2007.

Vassiliadis, P., & Simitsis, A. (2009). Extraction-Transformation-Loading. In *Encyclopedia of Database Systems*, L. Liu, T.M. Özsu (eds), Springer, 2009.

Vassiliadis, P., Karagiannis, A., Tziovara, V., & Simitsis, A. (2007). *Towards a Benchmark for ETL Workflows*. 5th International Workshop on Quality in Databases (QDB 2007), held in conjunction with VLDB 2007, Vienna, Austria, 23 September 2007.

Vassiliadis, P., Quix, C., Vassiliou, Y., & Jarke, M. (2001). Data Warehouse Process Management. *Information Systems*, *26*, 3, pp. 205-236

Vassiliadis, P., Simitsis, A., & Skiadopoulos, S. (2002). *Modeling ETL activities as graphs*. In Proc. 4th International Workshop on the Design and Management of Data Warehouses (DMDW 2002), held in conjunction with the 14th Conference on Advanced Information Systems Engineering (CAiSE'02), pp. 52-61, Toronto, Canada, May 27, 2002.

Vassiliadis, P., Simitsis, A., & Skiadopoulos, S. (2002). *Conceptual Modeling for ETL Processes.* In Proc. ACM 5th International Workshop on Data Warehousing and OLAP (DOLAP 2002), McLean, VA, USA November 8, 2002.

Vassiliadis, P., Simitsis, A., Georgantas, P., & Terrovitis, M. (2003). *A Framework for the Design of ETL Scenarios*. In Proc. 15th Conference on Advanced Information Systems Engineering (CAiSE 2003), pp. 520- 535, Klagenfurt/Velden, Austria, 16 - 20 June, 2003.

Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., & Skiadopoulos, S. (2005). A generic and customizable framework for the design of ETL scenarios. *Information Systems*, *30*, 7, 492-525.

Vassiliadis, P., Simitsis, A., Terrovitis, M., & Skiadopoulos, S. (2005). *Blueprints for ETL workflows*. In Proc. 24th International Conference on Conceptual Modeling (ER 2005), pp. 385-400, 24-28 October 2005, Klagenfurt, Austria. Long version available at

http://www.cs.uoi.gr/~pvassil/publications/2005_ER_AG/ETL_blueprints_long.pdf (last accessed at 25 July 2008)

Panos Vassiliadis received the PhD degree from the National Technical University of Athens in 2000. Since 2002, he has been with the Department of Computer Science, University of Ioannina, Greece, where he is also a member of the Distributed Management of Data (DMOD) Laboratory (http://www.dmod.cs.uoi.gr). His research activity and published work concerns the area of data warehousing, with particular emphasis on metadata, OLAP, and ETL, as well as the areas of database evolution and web services. He is a member of the ACM, the IEEE, and the IEEE Computer Society.