EXTRACTION, TRANSFORMATION, AND LOADING

Panos Vassiliadis
University of Ioannina
Ioannina, Hellas
pvassil@cs.uoi.gr

Alkis Simitsis
IBM Almaden Research Center
San Jose CA, 95120, USA
asimits@us.ibm.com

SYNONYMS

ETL; ETL process; ETL tool; Back Stage of a Data Warehouse; Data warehouse refreshment

DEFINITION

Extraction, Transformation, and Loading (ETL) processes are responsible for the operations taking place in the back stage of a data warehouse architecture. In a high level description of an ETL process, first, the data are extracted from the source datastores, which can be in a relational and/or a semi-structured format. In typical cases, the source datastores can be On-Line Transactional Processing (OLTP) or legacy systems, files under any format, web pages, various kinds of documents (e.g., spreadsheets and text documents) or even data coming in a streaming fashion. Typically, only the data that are different from the previous execution of an ETL process (newly inserted, updated, and deleted information) should be extracted from the sources. After this phase, the extracted data are propagated to a special-purpose area of the warehouse, called Data Staging Area (DSA), where their transformation, homogenization, and cleansing take place. The most frequently used transformations include filters and checks to ensure that the data propagated to the warehouse respect business rules and integrity constraints, as well as schema transformations that ensure that data fit the target data warehouse schema. Finally, the data are loaded to the central data warehouse (DW) and all its counterparts (e.g., data marts and views). In a traditional data warehouse setting, the ETL process periodically refreshes the data warehouse during idle or low-load, periods of its operation (e.g., every night) and has a specific time-window to complete. Nowadays, business necessities and demands require near real-time data warehouse refreshment and significant attention is drawn to this kind of technological advancement.

HISTORICAL BACKGROUND

Despite the fact that ETL took its name and separate existence during the first decade of the 21st century, ETL processes have been a companion to database technology for a lengthier period of time –in fact, from the beginning of its existence. During that period, ETL software was just silently hidden as a routine programming task without any particular name or individual importance. ETL was born on the first day that a programmer constructed a program that takes records from a certain persistent file and populates or enriches another file with this information. 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. Apart from this low-profile programming task, research efforts have long hidden ETL tasks, although not much attention was paid to them.

The earliest form of ETL system that we know of goes back to the EXPRESS system [13] that was intended to act as an engine that produces data transformations given some data definition and conversion nonprocedural statements. In later years, during the early days of data integration, the driving force behind data integration were wrapper-mediator schemes; the construction of the wrappers is a primitive form of ETL scripting [12]. In the mid '90's, data warehousing came in the central stage of database research and still, ETL was there, but hidden behind the lines. Popular books [4] do not mention the ETL triplet at all, although the different parts (transformation, cleansing, staging of intermediate data, and loading) are all covered - even if this is done very

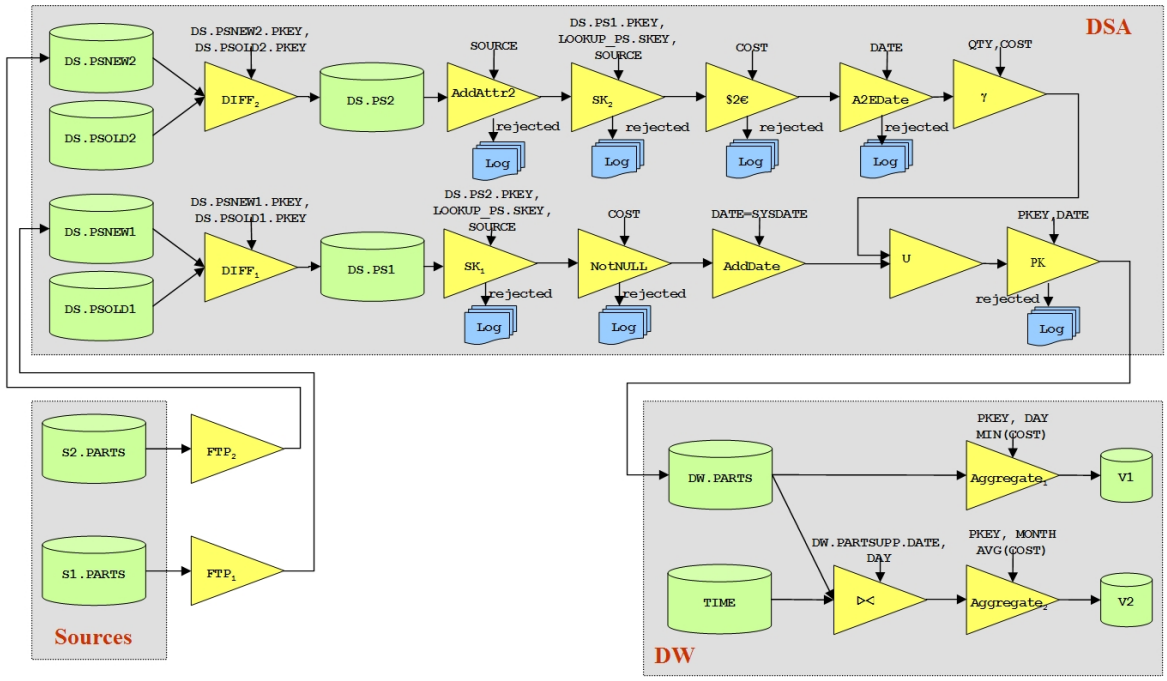

Figure 1: An example ETL workflow

briefly at times (it is also noteworthy that the 3rd edition of the same book in 2003 mentions ETL, although briefly). At the same time, early research literature treated data warehouses as collections of materialized views since this abstraction was simple and quite convenient for the formulation of research problems.
During the '00s, it became more and more prevalent that ETL is really important for data integration tasks since it is costly, labor-intensive, and mission-critical – and for all these factors, important for the success of a data warehousing project. The difficulty lies in the combination of data integration and data cleaning tasks: as a typical integration problem, where data are moved and transferred from a certain data store to another, the details of schema and value mappings play a great role. At the same time, the data can be in highly irregular state (both in terms of duplicates, constraint and business rule violations, and in terms of irregularities in the internal structure of fields – e.g., addresses). Therefore, finding effective ways for taming their non-regularity into a typical relational structure is very hard. On top of that, there is also a variety of persisting problems: ETL processes are hard to standardize, optimize and execute in a failure-resilient manner (and thus the problem is hard to solve, and worth paying good money for it).

SCIENTIFIC FUNDAMENTALS
PART I. General description of an ETL Process

Intuitively, an ETL process can be thought of as a directed acyclic graph, with activities and record sets being the nodes of the graph and input-output relationships between nodes being the edges of the graph. Observe Figure 1, where two sources are depicted in the lower left part of the figure with the names $S_1.PARTS$ and $S_2.PARTS$. The data from these sources must be propagated to the target data warehouse fact table $DW.PARTS$, depicted at the bottom right part of the figure. Moreover, the newly inserted records must be further propagated for the refreshment of aggregate views V_1 and V_2. (In fact, the views of the example can be anything among materialized views that are populated by the ETL process, materialized views automatically refreshed by the DBMS, convenient abstractions of data marts, reports and spreadsheets that are placed in the enterprise portal for the end-users to download, and any other form of aggregated information that is published in some form to the end-users of the warehouse.) The whole process of populating the fact table and any of its views is facilitated

by a *workflow of activities* that perform all the appropriate filtering, intermediate data staging, transformations and loadings. The upper part of Figure 1 depicts how the data (which are extracted as snapshots of the sources) are transported to a special purpose intermediate area of the warehouse, called Data Staging Area (DSA) [5], where the transformation phase takes place. First, the snapshots are compared to their previous versions, so that newly inserted or updated data are discovered. Then, these new data are stored in a hard disk so that we avoid starting the whole process from scratch in the case of a failure. Then, the data pass from several filters or transformations and they are ultimately loaded in the data warehouse. The warehouse fact or dimension tables are not necessarily the end of the journey for the data: at the bottom right of Figure 1, a couple of materialized views (standing as abstractions of reports, spreadsheets or data marts for the sake of the example) are also refreshed with newly incoming data.

PART II. Individual steps

Extraction. The extraction step is conceptually the simplest task of all, with the goal of identifying the correct subset of source data that has to be submitted to the ETL workflow for further processing. As with the rest of the ETL process, extraction also takes place at idle times of the source system - typically at night. Practically, the task is of considerable difficulty, due to two technical constraints:

- the source must suffer minimum overhead during the extraction, since other administrative activities also take place during that period, and,
- both for technical and political reasons, administrators are quite reluctant to accept major interventions to their system's configuration; therefore, there must be minimum interference with the software configuration at the source side.

Depending on the technological infrastructure and the nature of the source system (relational database, COBOL file, spreadsheet, web site etc.) as well as the volume of the data that has to be processed, different policies can be adopted for the extraction step, which usually is also called "change data capture". The most naïve possibility involves extracting the whole source and processing it as if the original first loading of the warehouse was conducted. A better possibility involves the extraction of a snapshot of data, which is subsequently compared to the previous snapshot of data (either at the source, or the DSA side) and insertions, deletions and updates are detected. In this case, there is no need to further process the data that remain the same. The work presented in [6] is particularly relevant in this context. Another possibility, involves the usage of triggers in the source that are activated whenever a modification takes place in the source database. Obviously, this can be done only if the source database is a relational system; most importantly though, both the interference with the source system and the runtime overhead incurred are rather deterring factors with respect to this option. An interesting possibility, though, involves the "log sniffing", i.e., the appropriate parsing of the log file of the source. In this case, all modifications of committed transactions are detected and they can be "replayed" at the warehouse side. A final point in the extraction step involves the necessity of encrypting and compressing the data that are transferred from the source to the warehouse, for security and network performance reasons, respectively.

Transformation. Depending on the application and the tool used, ETL processes may contain a plethora of transformations. In general, the transformation and cleaning tasks deal with classes of conflicts and problems that can be distinguished in two levels [8]: the schema and the instance level. We apply a broader classification of the problems that includes value-level problems, too.

- *Schema-level problems.* The main problems with respect to the schema level are (a) naming conflicts, where the same name is used for different objects (homonyms) or different names are used for the same object (synonyms) and (b) structural conflicts, where one must deal with different representations of the same object in different sources, or converting data types between sources and the warehouse.
- *Record-level problems.* The most typical problems at the record level concern duplicated or contradicting records. Furthermore, consistency problems concerning the granularity or timeliness of data occur, since the designer is faced with the problem of integrating data sets with different aggregation levels (e.g., sales per day vs. sales per year) or reference to different points in time (e.g., current sales as of yesterday for a certain source vs. as of last month for another source).

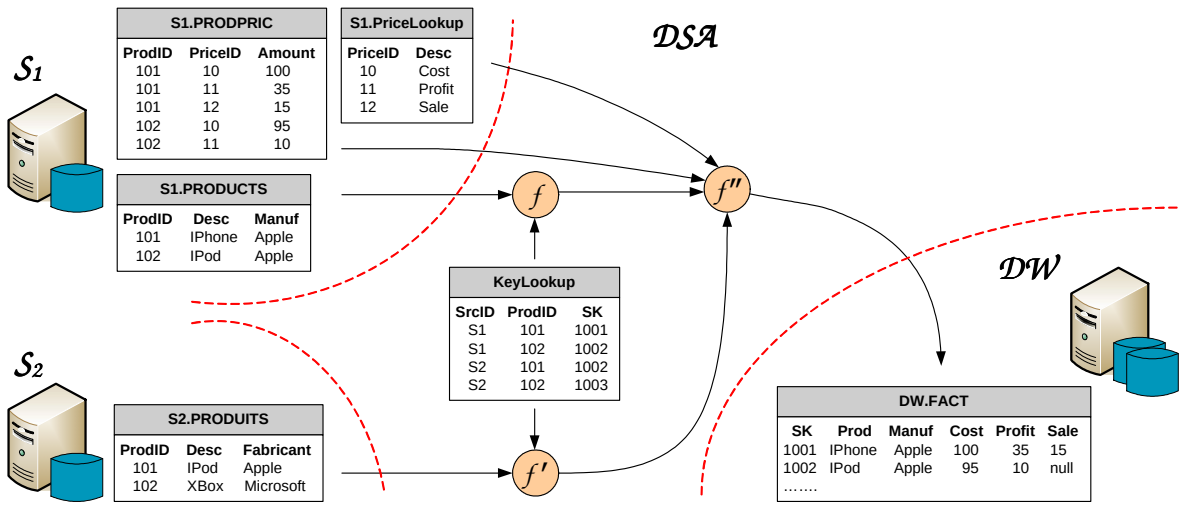

Figure 2: Pivot and Surrogate Keys Assignment operators

- *Value-level problems.* Finally, numerous low-level technical problems may be met in different ETL scenarios. To mention a few, there may exist problems in applying format masks, like for example, different value representations (e.g., for sex: 'Male', 'M', '1'), or different interpretation of the values (e.g., date formats: American 'mm/dd/yy' vs. European 'dd/mm/yy'). Other value-level problems include assigning surrogate key management (see next), substituting constants, setting values to NULL or DEFAULT based on a condition, or using frequent SQL operators like UPPER, TRUNC, SUBSTR.

To deal with such issues, the integration and transformation tasks involve a wide variety of functions, such as normalizing, denormalizing, reformatting, recalculating, summarizing, merging data from multiple sources, modifying key structures, adding an element of time, identifying default values, supplying decision commands to choose between multiple sources, and so forth.

To give a better picture of the problems that the designer of an ETL process usually meets, we elaborate on four frequent problems in a data warehouse environment.

Surrogate Keys. A common technique in a data warehouse is to replace the keys of the production systems with a uniform key, namely surrogate key. This replacement is due to a twofold reason: performance and semantic homogeneity. The performance is impacted when the production keys in the sources follow a textual type, which is not appropriate for indexing; then, a good policy is to replace the textual key values with numerical values. However, the most crucial issue is the semantic homogeneity. Frequently, different production systems use different keys for the same object (synonyms), or the same key for different objects (homonyms), resulting in the need for a global replacement of those values in the data warehouse. To better illustrate this problem, observe Figure 2. In that example, two sources S_1 and S_2 populate the fact table $DW.FACT$ of the data warehouse. The table $S_1.PRODUCTS$ contains the tuple 102 that describes the product $IPod$ manufactured by $Apple$. This row is in conflict with row '$101, IPod, Apple$', in table $S_2.PRODUCTS$, because they both represent the same real-world entity with different ID's. In addition, the same row has a homonym conflict with row '$102, XBox, Microsoft$', in table $S_2.PRODUCTS$ (over attribute $ProdID$). A solution to such problem is to replace the production key $ProdID$ by a surrogate key using the lookup table $KeyLookup(SrcID, ProdID, SK)$. The information about the source (attribute $SrcID$) allows the existence of synonyms in different sources that are mapped to different objects in the data warehouse (e.g., value 102 in tables $S_1.PRODUCTS$ and $S_2.PRODUCTS$). At the end of this process, the fact table $DW.FACT$ has globally unique, reconciled keys.

Pivoting and Un-Pivoting. A basic characteristic of the data warehouse and a main difference from a relational database is the high degree of denormalization and redundancy. This 'problematic' (in relational terms) design

is proven extremely efficient for certain types of querying; e.g., aggregate queries over large volumes of data. Consider, again, the example depicted in Figure 2. In order to estimate the total price of a product, in the source S_1 an aggregation is needed in table $S_1.PRODPRIC$ over $ProdID$, while in the table $DW.FACT$ a simple algebraic calculation suffices ($Cost + Profit - Sale$). The gain of this operation becomes clearer when an aggregation over the total price of a huge number of products is required. Therefore, a very useful ETL operation is the transformation of a schema based on attribute-value pairs to a (possibly denormalized) schema appropriate for a data warehouse, where the coded values are mapped to named attributes. In other words, mostly due to reporting queries, the transformation of the information organized in rows, even with the price of denormalization, is favored over normalized information organized in columns. This type of transformation is so called *pivoting*, rotation or denormalization, because, frequently, the derived values (e.g., the total price) are also stored as columns, functionally dependent on other attributes. The reverse transformation is also applicable to several cases; i.e., to normalize denormalized data before being loaded to the data warehouse. As a final remark, observe in Figure 2, how the different tables at the source side (e.g., $PRODUCTS$, $PRODPRIC$, $PriceLookup$) are integrated into a single warehouse table (i.e., $DW.FACT$). Nowadays, several ETL tools support this operation as a built-in function. A study on this operation can be found in [1].

Slowly Changing Dimensions. During the refreshment of data warehouses with new values, the values of the dimension tables can change at the sources (e.g., a product's color might change) and whenever this happens, there are several policies to apply for their propagation to the data warehouse. The available solutions constitute the Slowly Changing Dimensions (SCD) pattern, and the three policies of [5], concerning the management of new and old values are: overwrite the old value (SCD Type 1), create a new dimensional record (SCD Type 2), push down the changed value into an "old" attribute (SCD Type 3).
In the Type 1 policy, the attribute values in existing dimension table tuples are simply overwritten by update commands. The procedure is realized as follows. Two snapshots of the dimensional data containing the current and the previous versions of the data, respectively, are compared. The newly inserted rows are assigned a new surrogate key and are inserted in the dimension. The updated rows are assigned a surrogate key, which is the same as the one that their previous version had already being assigned. Next, these rows are joined with their old versions from the dimension table, and the produced tuple is inserted in the target table (the old tuple is deleted). This policy is used when tracking history in dimensions changes is not necessary and the correction of values is needed.
In the Type 2 policy, the previous version of the dimension tuple is copied, if it exists, and a new tuple is created with a new surrogate key. An extra key linking the different variants of the same real-world entity is used (possibly in the form of a timestamp-based key). An annotation of the most recent variant as such (either through an extra, indexed attribute, or through a different table) proves to be very useful for efficiency reasons. This policy is used when tracking history of dimension changes is needed.
In the Type 3 policy, the attribute values in existing dimension table tuples are modified by ad-hoc update commands. If an attribute a_i is checked for updates, then a new attribute a_i_old is created. If a new value exist for a_i in a certain tuple, then the old value is copied to the attribute a_i_old and the new value is written in a_i. Hence, both values are kept in the respective tuple. This policy too is useful when tracking history of dimension changes is needed.

String Problems. Last but not least, a crucial problem in ETL is the cleaning and the homogenization of textual data. Such data may contain information about names, acronyms, addresses, dates, and so on. For example, consider the following case: the same entity that represents the country 'Greece' may be met by the following semantically equivalent (in real-world terms) expressions: 'Greece', 'GR', 'Hellas', etc. Such problems are extremely difficult to be handled. The numerous and usually quite expensive approaches that may be followed, include the usage of thesaurus or domain ontologies, and the application of regular expressions for the normalization of string data to a set of reference values.

Loading. The end of the source records' journey through the ETL workflow comes with their loading to the appropriate table. A typical dilemma faced by inexperienced developers concerns the choice between bulk loading data through a DBMS-specific utility or inserting data as a sequence of rows. Clear performance reasons strongly suggest the former solution, due to the overheads of the parsing of the insert statements, the maintenance of logs

and rollback-segments (or, the risks of their deactivation in the case of failures). A second issue has to do with the possibility of efficiently discriminating records that are to be inserted for the first time, with records that act as updates to previously loaded data. DBMS's typically support some declarative way to deal with this problem (e.g., Oracle's MERGE command [10]). In addition, simple SQL commands are not sufficient since the 'open-loop-fetch' technique, where records are inserted one by one, is extremely slow for the vast volume of data to be loaded in the warehouse. A third performance issue that has to be taken into consideration by the administration team has to do with the existence of indexes, materialized views, or both, defined over the warehouse relations. Every update to these relations automatically incurs the overhead of maintaining the indexes and the materialized views.

Part III. Global picture revisited

Design and Modeling. After the above analysis, one can understand that the design of ETL processes is crucial and complex at the same time. There exist several difficulties underneath the physical procedures already described.

At the beginning of a data warehousing project, the design team in cooperation with the business managers have to clarify the requirements, identify the sources and the target, and determine the appropriate transformations for that specific project; i.e., the first goal is to construct a *conceptual design of the ETL process* [17, 16, 9]. The most important part of this task is to identify the schema mappings between the source and the target schemata. For example, in Figure 2, one should identify the mapping between S_1, whose metadata are in English, and S_2, whose metadata are in French; e.g., both tables $PRODUCTS$ and $PRODUITS$ conceptually represent the same kind of information. This procedure is usually done through analysis of the structure and content of the existing data sources and their intentional mapping to the common data warehouse model. Possible data cleaning problems can also be detected at this early part of the design. Conceptual modeling formalisms (both ad-hoc and UML-based) have been proposed in the literature [17, 16, 9]. Also, several research approaches have been proposed towards the automation of the schema mapping procedure [11]; here, we mention CLIO, the most prominent one, which has also been adapted by commercial solutions proposed by IBM [3].

When this task ends, the design involves the *specification of a primary data flow* that describes the route of data from the sources towards the data warehouse, as they pass through the transformations of the workflow. The execution sequence of the workflow's transformation is determined at this point. The data flow defines what each process does and the execution plan defines in which order and combination. The flow for the logical exceptions – either integrity, or business rules violations is specified, too. Control flow operations that take care of monitoring, or failure occurrences can also be specified. Most ETL tools provide the designer with the functionality to construct both the data and the control flow for an ETL process.

Implementation and Execution. An ETL workflow can either be built in-house, or can be specified and executed via an ETL tool. The in-house implementation is a frequent choice, both due to the cost of ETL tools and due to the fact that developers feel comfortable to implement the scripts that manipulate their data by themselves. Typically, such an ETL workflow is built as the combination of scripts written in some procedural languages with high execution speed (for example, C or Perl), or some vendor specific database language (PL\SQL, T-SQL, etc.). Alternatively, ETL tools are employed, mainly due to the graphical programming interfaces they provide as well as for their reporting, monitoring, recovery facilities. Currently, all major DBMS vendors provide ETL functionality; products from other vendors are also in the market. Wikipedia provides a starting point for a list of ETL tools [18].

Optimization. A typical ETL workflow involves numerous transformations applied on large volumes of data stemming from different sources and populating different target relations. We can construct such a workflow in more than one manner. Observe the example of Figure 2. In this case, one has to decide (a) which of the three transformations f, f' or f'' should precede in terms of execution order and (b) what will the physical implementation of each of these transformations be.
Frequently, an ETL workflow must be completed in a specific time window and this task is realized periodically; e.g., each night. In the case of a failure, the quick recovery of the workflow is also important [7]. Hence, for performance reasons, it is necessary to *optimize the workflow's execution time*. Currently, although the leading

commercial tools provide advanced GUI's for the design of ETL scenarios, they do not support the designer with any optimization technique to the created scenarios. Unlike relational querying, where the user expresses a query in a high-level language and the DBMS optimizer decides the physical execution of the query automatically, in the case of ETL workflows it is the designer who must decide the order and physical implementation for the individual activities.
Practical alternatives involve (a) letting the involved DBMS (in the case of DBMS-based scripts) do the optimization and (b) treating the workflow as a big multi-query. The solution where optimization is handed over to the DBMS for execution is simply not sufficient, since DBMS optimizers can interfere only in portions of a scenario and not in its entirety. Concerning the latter solution, it should be stressed that *ETL workflows are not big queries*, since:

- It is not possible to express all ETL operations in terms of relational algebra and then optimize the resulting expression as usual. In addition, the cases of functions with unknown semantics - 'black-box' operations – or with 'locked' functionality – e.g., an external call to a DLL library – are quite often.
- Failures are a critical danger for an ETL workflow. The staging of intermediate results is often imposed by the need to resume a failed workflow as quickly as possible.
- ETL workflows may involve processes running in separate environments, usually not simultaneously and under time constraints; thus their cost estimation in typical relational optimization terms is probably too simplistic.

All the aforementioned reasons can be summarized by mentioning that neither the semantics of the workflow can always be specified, nor its structure can be determined solely on these semantics; at the same time, the research community has not come-up with an accurate cost model so far. Hence, it is more realistic to consider ETL workflows as complex transactions rather than as complex queries. Despite the significance of such optimization, so far the problem has not been extensively considered in research literature, with results mainly focused on the black-box optimization at the logical level, concerning the order with which the activities are placed in the ETL workflow [14].

KEY APPLICATIONS

ETL processes constitute the backbone of the data warehouse architecture. The population, maintenance, evolution and freshness of the warehouse heavily depends on its backstage where all the ETL operations are taken place. Hence, in a corporate environment there is a necessity for a team devoted to the design and maintenance of the ETL functionality.
However, ETL is not useful only for the refreshment of large data warehouses. Nowadays, with the advent of Web 2.0 new applications has been emerged. Among them, mashups are web applications that integrate data which are dynamically obtained via web-service invocations to more than one sources into an integrated experience. Example applications include Yahoo Pipes (http://pipes.yahoo.com/), Google Maps (http://maps.google.com/), IBM Damia (http://services.alphaworks.ibm.com/damia/), and Microsoft Popfly (http://www.popfly.com/). Under the hood, the philosophy for their operation is 'pure' ETL. Although, the extraction phase mainly contains functionality that allows the communication with web pages (just one more type of source), the transformations that constitute the main flow resemble those which are already built-in in most ETL tools. Different applications are targeting to gather data from different users, probably in different formats, and try to integrate them into a common repository of datasets; an example application is the Swivel (http://www.swivel.com/).

FUTURE DIRECTIONS

Although, as discussed, the ETL logic is not novel in computer science, still, several issues remain open (and not only from a research point of view). A main open problem in the so called traditional ETL is the agreement upon a unified algebra and/or a declarative language for the *formal description of ETL processes*. There are some first results in the context of the data exchange problem [2], but still, there is much work to do towards providing ETL engines with the equivalent of what relational algebra and SQL constitute for DBMSs. As already mentioned, the *optimization* of the whole ETL process, but also of any individual transformation operators pose interesting research problems. In this context, parallel processing of ETL processes is of particular importance.

Finally, *standardization* is a problem that needs an extra note of attention: although several commercial tools already exist in the market and all major DBMS vendors provide ETL functionality, each individual effort follows a different approach for the modeling and representation of ETL processes and individual components. The convergence towards a globally accepted paradigm of thinking end educating computer scientists on the topic is a clear issue for the academic community.

However, the ETL functionality expands into new areas beyond the traditional data warehouse environment, where the ETL is executed off-line, on a regular basis.

On-Demand ETL processes are executed sporadically, and they are manually initiated by some user demand. The process is responsible for retrieving external data and loading them in the warehouse after the appropriate transformations. Interesting problems in this area contain: the need for specialized operators, since this process is mostly focused towards Web data, the computation of minimum effort/time/resources for the construction of the process; the need for efficient algorithms, due to the fact that this process is initiated by the user; the provision of a framework easily adaptable to the changes of the external data, and so on.

Stream ETL involves the possible filtering, value conversion, and transformations of incoming streaming information in a relational format. Although, streams cannot be stored, some patterns or snapshot aggregates of them can be stored for subsequent querying. Several interesting issues can be identified in this area: the provision of correctness guarantees; the necessity of cost models for the tuning of the incoming stream within specified time window; the audit of the incoming stream data for several constraints or business rules, also with respect to stored data (e.g., primary key violations).

Recently, the term *active warehousing* was coined to capture the need for a data warehouse containing data as fresh as possible. Modern corporate necessities demand that OLAP and DSS (Decision support Systems) should handle updated data in an on-line fashion. The implementation of workflows that transfer data from the sources to the warehouse immediately as they are generated or modified currently seems to be impossible, due to the overloading of both the operational sources and the warehouse. Therefore, the path towards achieving this functionality seems to be a significant increase of the frequency with which the ETL process is performed (again, having system loading, performance and data completeness issues in mind). Approximation and streaming techniques should be taken into consideration too to serve user needs with approximate information. In any case, it appears that currently (2007) active ETL is an established necessity and traditional ETL should adapt to a new on-line environment.

Finally, with the evolution of the technology and the broader use of internet from bigger masses of users, the interest is moved also *to multiple types of data*, which do not necessarily follow the traditional relational format. Thus, modern ETL applications should also handle efficiently XML, spatial, biomedical, multimedia data and so on.

DATA SETS

Benchmarking the ETL process is a clear problem nowadays (2007). The lack of any standard principled experimental methodology with a clear treatment of the workflow complexity, the data volume, the amount of necessary cleaning and the computational cost of individual activities of the ETL workflows is striking. The only available guidelines for performing a narrow set of experiments are given in the TPC-DS standard [15].

CROSS REFERENCE

Data Cleaning, Data Warehouse

RECOMMENDED READING

[1] C. Cunningham, G. Graefe, and C. A. Galindo-Legaria. Pivot and unpivot: Optimization and execution strategies in an rdbms. In *VLDB*, pages 998–1009, 2004.

[2] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.

[3] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD Conference*, pages 805–810, 2005.

[4] W. Inmon. *Building the Data Warehouse*. John Wiley & Sons, 2nd edition, 1996.

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

[6] W. Labio and H. Garcia-Molina. Efficient snapshot differential algorithms for data warehousing. In *VLDB*, pages 63–74, 1996.

[7] W. Labio, J. L. Wiener, H. Garcia-Molina, and V. Gorelik. Efficient resumption of interrupted warehouse loads. In *Proceedings of ACM SIGMOD*, pages 46–57. ACM, 2000.

[8] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[9] S. Luján-Mora, P. Vassiliadis, and J. Trujillo. Data mapping diagrams for data warehouse design with UML. In *23rd International Conference on Conceptual Modeling (ER 2004)*, pages 191–204, 2004.

[10] Oracle. *Oracle9i SQL Reference. Release 9.2*, 2002.

[11] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[12] M. T. Roth and P. M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *VLDB*, pages 266–275, 1997.

[13] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. Express: A data extraction, processing, amd restructuring system. *ACM Trans. Database Syst.*, 2(2):134–174, 1977.

[14] A. Simitsis, P. Vassiliadis, and T. K. Sellis. State-space optimization of ETL workflows. *IEEE Transactions on Knowledge and Data Engineering*, 17(10):1404–1419, 2005.

[15] TPC. *TPC-DS (Decision Support) specification, draft version 52*, February 2007.

[16] J. Trujillo and S. Luján-Mora. A UML based approach for modeling etl processes in data warehouses. In *22nd International Conference on Conceptual Modeling (ER 2003)*, pages 307–320, 2003.

[17] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for ETL processes. In *Proceedings of the ACM 5th International Workshop on Data Warehousing and OLAP (DOLAP '02)*, pages 14–21, 2002.

[18] Wikipedia. ETL tools. Technical report, Availaible at http://en.wikipedia.org/wiki/Category:ETL_tools, 2007.

EXTRACTION, TRANSFORMATION, AND LOADING

BYLINE
(Name of expert writing the entry, expert's affiliation)

SYNONYMS
synonym1; synonym2; etc.

DEFINITION
250 or fewer words defining the entry title.

MAIN TEXT
250 words outlining the key points.

CROSS REFERENCE
Listing of related entries, e.g. Remote Sensing, Spatial Data Quality, etc.

REFERENCES
Optional. A list of 1-3 citations that give the reader a place to find more information.