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; Data warehouse back stage; 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 data stores that can be On-Line Transaction 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 the 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 obtained its 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.

The earliest form of ETL system 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 [3] 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 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.

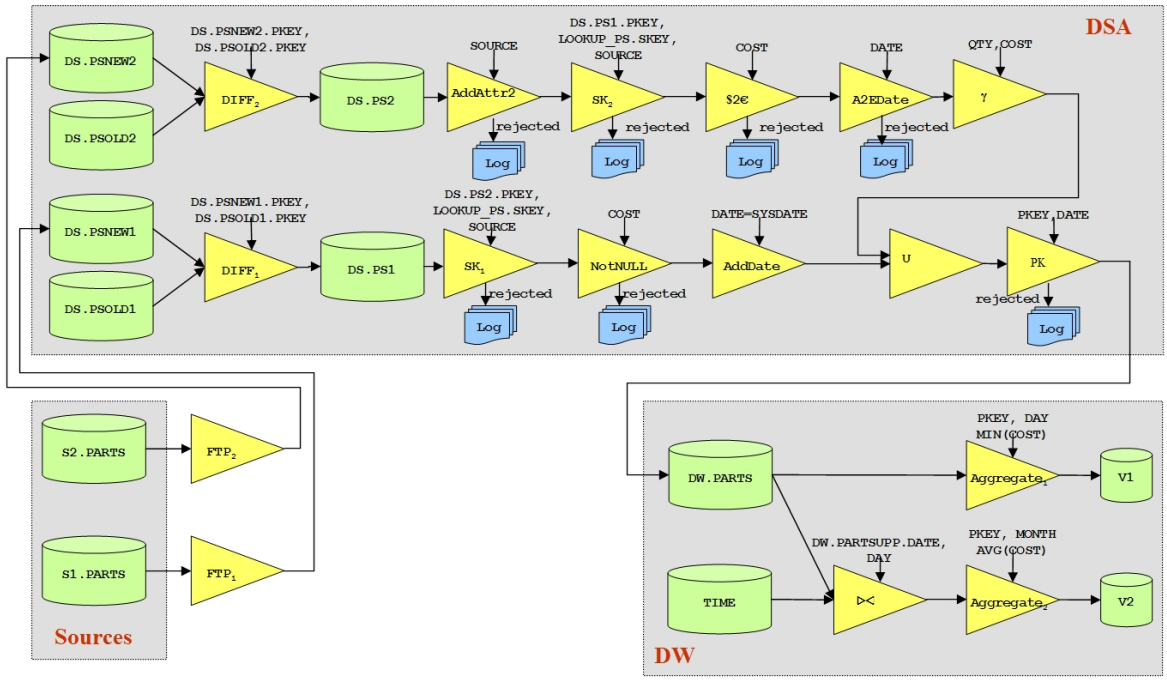

Figure 1: An example ETL workflow

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 this 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).

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 the Data Staging Area (DSA)

[4], where the transformation phase takes place. First, the snapshots are compared to their previous versions, so that newly inserted or updated data are discovered. Next, these new data are stored in a hard disk so that in the case of a failure, the whole process should not start from scratch. Then, the data pass from several filters or transformations and they are ultimately loaded in the data warehouse. As already mentioned, 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 [5] 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 [7]: the schema and the instance level. In this article, a broader classification of the problems is presented 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).
- *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, substituting constants, setting values to NULL or DEFAULT based on a condition, or using frequent SQL operators like UPPER, TRUNC, and 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.

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, from 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 [9]). 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* [20, 18, 8]. The most important part of this task is to identify the schema mappings between the source and the target schemata. 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 [20, 18, 8]. Also, several research approaches have been proposed towards the automation of the schema mapping procedure [10]; one of the most prominent examples is CLIO, which has also been adapted by commercial solutions proposed by IBM [2]. However, so far, the automatic schema mapping supports cases of simple ETL transformations. On the contrary, there exists a semi-automatic method that uses ontologies and semantic web technology to infer the appropriate interattribute mappings along with the respective transformations needed in an ETL process [16].

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.

Optimization. Usually, 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 [6]. 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 technique to optimize the created scenarios. Unlike relational querying, where the user declaratively 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 common.
- 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, 15].

Implementation and Execution. An ETL workflow can either be hand-coded, or can be specified and executed via an ETL tool. The hand-coded 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, and so on). Alternatively, ETL tools are employed, mainly due to the graphical programming interfaces they provide as well as for their reporting, monitoring, and recovery facilities.

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 have 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 over the web, 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, the ETL logic is not novel in computer science, still, several issues remain open. 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*. However, there are some first results in the context of the data exchange problem [1]. 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. The convergence toward a globally accepted paradigm of thinking and 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. Such cases include but are not limited to: (a) *On-Demand ETL processes* that are executed sporadically (typically for Web data), and they are manually initiated by some user demand [11]; (b) *Stream ETL* that involves the possible filtering, value conversion, and transformations of incoming streaming information in a relational format [11]; (c) *(near)Real-Time ETL* that captures the need for a data warehouse containing data as fresh as possible.

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 novel kinds of data, like XML, spatial, biomedical, or multimedia data efficiently.

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 [17], and the first publicly available benchmark for ETL processes is presented in [19].

CROSS REFERENCE

Data Cleaning, Data Warehouse, Active Data Warehousing, (near)Real-Time Data Warehousing, Multidimensional Modeling

RECOMMENDED READING

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

[2] 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.

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

[4] 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.

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

[6] 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.

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

[8] 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.

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

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

[11] S. Rizzi, A. Abelló, J. Lechtenbörger, and J. Trujillo. Research in data warehouse modeling and design: dead or alive? In *DOLAP '06: Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, pages 3–10, 2006.

[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. Optimizing ETL processes in data warehouses. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*, pages 564–575, 2005.

[15] 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.

[16] D. Skoutas and A. Simitsis. Designing ETL processes using semantic web technologies. In *Proceedings of the ACM 9th International Workshop on Data Warehousing and OLAP (DOLAP '06)*, pages 67–74, 2006.

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

[18] 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.

[19] P. Vassiliadis, A. Karagiannis, V. Tziovara, P. Vassiliadis, and A. Simitsis. Towards a Benchmark for ETL Workflows. In *5th International Workshop on Quality in Databases (QDB) at VLDB*, 2007.

[20] 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.