Benchmarking ETL Workflows

Alkis Simitsis[1], Panos Vassiliadis[2], Umeshwar Dayal[1],
Anastasios Karagiannis[2], Vasiliki Tziovara[2]

[1] HP Labs, Palo Alto, CA, USA,
{alkis, Umeshwar.Dayal}@hp.com
[2] University of Ioannina, Dept. of Computer Science, Ioannina, Hellas
{pvassil, ktasos, vickit}@cs.uoi.gr

Abstract. Extraction–Transform–Load (ETL) processes comprise complex data workflows, which are responsible for the maintenance of a Data Warehouse. A plethora of ETL tools is currently available constituting a multi-million dollar market. Each ETL tool uses its own technique for the design and implementation of an ETL workflow, making the task of assessing ETL tools extremely difficult. In this paper, we identify common characteristics of ETL workflows in an effort of proposing a unified evaluation method for ETL. We also identify the main points of interest in designing, implementing, and maintaining ETL workflows. Finally, we propose a principled organization of test suites based on the TPC-H schema for the problem of experimenting with ETL workflows.

Keywords: Data Warehouses, ETL, benchmark.

1 Introduction

Data warehousing is a technology that enables decision-making and data analysis in large organizations. Several products are available in the market and for their evaluation, the TPC-H benchmark has been proposed as a decision support benchmark [16]. TPC-H focuses on OLAP (On-Line Analytical Processing) queries and it mainly deals with the data warehouse site. Another version termed TPC-DS has been around for the last few years, but this version is still in a draft form [11, 15]. TPC-DS considers a broader picture than TPC-H including the whole flow from the sources to the target data warehouse. However, it partially covers the data warehouse maintenance part, considering only simple mechanisms for inserting and deleting tuples.

To populate a data warehouse with up-to-date records extracted from operational sources, special tools are employed, called *Extraction – Transform – Load* (ETL) tools, which organize the steps of the whole process as a workflow. To give a general idea of the functionality of these workflows we mention their most prominent tasks, which include: (a) the *identification* of relevant information at the source side; (b) the *extraction* of this information; (c) the *transportation* of this information to the Data Staging Area (DSA), where most of the transformation usually take place; (d) the *transformation*, (i.e., customization and integration) of the information coming from multiple sources into a common format; (e) the *cleansing* of the resulting data set, on the basis of database and business rules; and (f) the *propagation* and loading of the data to the data warehouse and the refreshment of data marts.

Due to their importance and complexity (see [2, 12] for relevant discussions and case studies), ETL tools constitute a multi-million dollar market. There is a plethora of commercial ETL tools available. The traditional database vendors provide ETL solutions along with the DBMS's: IBM with InfoSphere Information Server [7], Microsoft with SQL Server Integration Services (SSIS) [9], and Oracle with Oracle Warehouse Builder [10]. There also exist independent vendors that cover a large part of the market (e.g., Informatica with Powercenter [8] and Ab Initio [1]). Nevertheless, an in-house development of the ETL workflow is preferred in many data warehouse projects, due to the significant cost of purchasing and maintaining an ETL tool. The spread of existing solutions comes with a major drawback. Each one of them follows a different design approach, offers a different set of transformations, and provides a different internal language to represent essentially similar functions.

Although Extract-Transform-Load (ETL) tools are available in the market for more than a decade, only in the last few years have researchers and practitioners started to realize the importance that the integration process has in the success of a data warehouse project. There have been several efforts towards (a) modeling tasks and the automation of the design process, (b) individual operations (with duplicate detection being the area with most of the research activity) and (c) some first results towards the optimization of the ETL workflow as a whole (as opposed to optimal algorithms for their individual components). For lack of space, we refer the interested reader to [12] for a detailed survey on research efforts in the area of ETL tools.

The wide spread of industrial and ad-hoc solutions combined with the absence of a mature body of knowledge from the research community is responsible for the absence of a principled foundation of the fundamental characteristics of ETL workflows and their management. A small list of shortages concerning such characteristics include: no principled taxonomy of individual activities exists, few efforts have been made towards the optimization of ETL workflows as a whole, and practical problems like recovering from failures and handling evolution have mostly been ignored. Thus, *a commonly accepted, realistic framework for experimentation is also absent.*

Contributions. In this paper, we aim at providing a principled categorization of test suites for the problem of experimenting with a broad range of ETL workflows. First, we provide a principled way for constructing ETL workflows (Section 2). We identify the main functionality provided by representative commercial ETL tools and categorize the ETL operations into abstract logical activities. Based on that, we propose a categorization of ETL workflows, which covers frequent design cases. Then, we describe the main configuration parameters and a set of measures to be monitored for capturing the generic functionality of ETL tools (Section 3). Finally, we provide specific ETL scenarios based on the aforementioned analysis, which can be used as an experimental testbed for the evaluation of ETL design methods or tools (Section 4).

2. Problem Formulation

In this section, we introduce ETL workflows as graphs. Then, we zoom in the *micro-level* of ETL workflows inspecting each individual activity in isolation and then, we return at the *macro-level*, inspecting how individual activities are "tied" altogether to compose an ETL workflow. Finally, we discuss the characteristics of ETL execution and we tie them to the goals of the proposed benchmark.

2.1 ETL workflows

An ETL workflow is a design blueprint for the ETL process. The designer constructs a workflow of activities (or operations), usually in the form of a graph, to specify the order of cleansing and transformation operations that should be applied to the source data, before being loaded to the data warehouse. In what follows, we use the term *recordsets* to refer to any data store that obeys a schema (such as relational tables and record files) and the term *activity* to refer to any software module that processes the incoming data, either by performing any schema transformation over the data or by applying data cleansing procedures. Activities and recordsets are *logical abstractions* of physical entities. At the logical level, we are interested in their schemata, semantics, and input-output relationships; however, we do not deal with the actual algorithm or program that implements the logical activity or with the storage properties of a recordset. When in a later stage, the logical-level workflow is refined at the physical level a combination of executable programs/scripts that perform the ETL workflow is devised. Then, each activity of the workflow is physically implemented using various algorithmic methods, each with different cost in terms of time requirements or system resources (e.g., CPU, memory, disk space, and disk I/O).

Formally, we model an ETL workflow as a directed acyclic graph $\mathbf{G}(\mathbf{V},\mathbf{E})$. Each node $v \in \mathbf{V}$ is either an activity a or a recordset r. An edge $(a,b) \in \mathbf{E}$ is a *provider relationship* denoting that b receives data from node a for further processing. Nodes a and b are the data *provider* and data *consumer*, respectively. The following well-formedness constraints determine the interconnection of nodes in ETL workflows:

- Each recordset r is a pair (*r.name, r.schema*), with the schema being a finite list of attribute names.
- Each activity a is a tuple (N,**I**,**O**,S,A). N is the activity's name. **I** is a finite set of input schemata. **O** is a finite set of output schemata. S is a declarative description of the relationship of its output schema with its input schema in an appropriate language (without delving into algorithmic or implementation issues). A is the algorithm chosen for activity's execution.
- The data consumer of a recordset cannot be another recordset. Still, more than one consumer is allowed for recordsets.
- Each activity must have at least one provider, either another activity or a recordset. When an activity has more than one data providers, these providers can be other activities or activities combined with recordsets.
- The data consumer of an activity cannot be the same activity.

2.2 Micro-level activities

At a micro level, we consider three broad categories of ETL activities: (a) *extraction* activities, (b) *transformation* and *cleansing* activities, and (c) *loading* activities.

Extraction activities extract the relevant data from the sources and transport them to the ETL area of the warehouse for further processing (possibly including operations like ftp, compress, etc.). The extraction involves either differential data sets with respect to the previous load, or full snapshots of the source. Loading activities have to deal with the population of the warehouse with clean and appropriately transformed

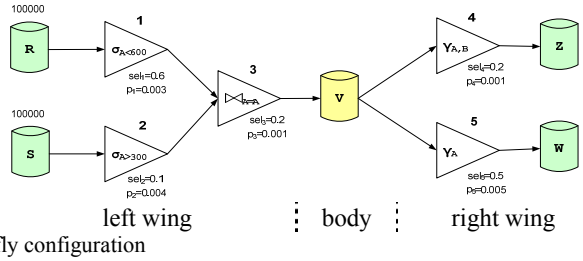


Figure 1. Butterfly configuration

data. This is typically done through a bulk loader program; nevertheless the process also includes the maintenance of indexes, materialized views, reports, and so on. Transformation and cleansing activities can be coarsely categorized with respect to the result of their application to data and the prerequisites, which some of them should fulfill. In this context, we discriminate the following categories of operations:

- *Row-level* operations, which are locally applied to a single row.
- *Router* operations, which locally decide, for each row, which of the many (output) destinations it should be sent to.
- *Unary Grouper* operations, which transform a set of rows to a single row.
- *Unary Holistic* operations, which perform a transformation to the entire data set. These are usually blocking operations.
- *Binary* or *N-ary* operations, which combine many inputs into one output.

All frequently built-in transformations in the majority of commercial solutions fall into our classification (see for example Figure A3 – in the appendix).

2.3 Macro level workflows

The macro level deals with the way individual activities and recordsets are combined together in a large workflow. The possibilities of such combinations are infinite. Nevertheless, our experience suggests that most ETL workflows follow several high-level patterns, which we present in a principled fashion in this section.

We introduce a broad category of workflows, called *Butterflies*. A butterfly (see also Figure 1) is an ETL workflow that consists of three distinct components: (a) the left wing, (b) the body, and (c) the right wing of the butterfly. The left and right wings (separated from the body with dashed lines in Figure 1) are two non-overlapping groups of nodes which are attached to the body of the butterfly. Specifically:

- The left wing of the butterfly includes one or more sources, activities and auxiliary data stores used to store intermediate results. This part of the butterfly performs the extraction, cleaning and transformation part of the workflow and loads the processed data to the body of the butterfly.
- The body of the butterfly is a central, detailed point of persistence that is populated with the data produced by the left wing. Typically, the body is a detailed *fact* or *dimension* table*;* still, other variants are also possible.
- The right wing gets the data stored at the body and utilizes them to support reporting and analysis activity. The right wing consists of materialized views, reports, spreadsheets, as well as the activities that populate them. In our setting, we abstract all the aforementioned static artifacts as materialized views.

Balanced Butterflies. A butterfly that includes medium-sized left and right wings is called a *Balanced butterfly* and stands for an ETL scenario where incoming source data are merged to populate a warehouse table along with several views or reports defined over it. Figure 1 is an example of this class of butterflies. This variant represents a symmetric workflow (there is symmetry between the left and right wings). However, this is not always the practice in real-world cases. For instance, the butterfly's triangle wings are distorted in the presence of a router activity that involves multiple outputs (e.g., copy, splitter, switch, and so on). In general, the two fundamental wing components can be either *lines* or *combinations*. In the sequel, we discuss these basic patterns for ETL workflows that can be further used to construct more complex butterfly structures. Figure 2 depicts example cases of these variants.

Lines. Lines are sequences of activities and recordsets such that all activities have exactly one input (unary activities) and one output. Lines form single data flows.

Combinations. A combinator activity is a join variant (a binary activity) that merges parallel data flows through some variant of a join (e.g., a relational join, diff, merge, lookup or any similar operation) or a union (e.g., the overall sorting of two independently sorted recordsets). A combination is built around a combinator with lines or other combinations as its inputs. We differentiate combinations as left-wing and right-wing combinations.

Left-wing combinations are constructed by lines and combinations forming the left wing of the butterfly. The left wing contains at least one combination. The inputs of the combination can be:

- *Two lines*. Two parallel data flows are unified into a single flow using a combination. These workflows are shaped like the letter '*Y*' and we call them *Wishbones*.
- *A line and a recordset*. This refers to the practical case where data are processed through a line of operations, some of which require a lookup to persistent relations. In this setting, the *Primary Flow* of data is the line part of the workflow.
- *Two or more combinations*. The recursive usage of combinations leads to many parallel data flows. These workflows are called *Trees*.

Observe that in the cases of trees and primary flows, the target warehouse acts as the body of the butterfly (i.e., there is no right wing). This is a practical situation that covers (a) fact tables without materialized views and (b) the case of dimension tables that also need to be populated through an ETL workflow. In some cases, the body of the butterfly is not necessarily a recordset, but an activity with many outputs (see last example of Figure 2). Then, the main goal of the scenario is to distribute data to the appropriate flows; this task is performed by an activity serving as the butterfly's body.

Right-wing combinations are created by lines and combinations on the right wing of the butterfly. These lines and combinations form either a flat or a deep hierarchy.

- *Flat Hierarchies*. These configurations have small depth (usually 2) and large fan-out. An example of such a workflow is a *Fork*, where data are propagated from the fact table to the materialized views in two or more parallel data flows.
- *Right - Deep Hierarchies*. We also employ configurations with *right-deep hierarchies*. These configurations have significant depth and medium fan-out.

A more detailed description of the above structures is given in Section 4.2.

Butterflies are important for benchmarking at least in the following ways. Since such constructs are based on the classification of ETL activities discussed before, they form a taxonomy as aid for designing or understanding complex ETL workflows. In

particular, we can use them for constructing more complex ETL workflows in a principle way. For example, if we need a memory intensive workflow, we should consider using tree or fork flows, which include routers/joins and a significant number of sorting or aggregating operations. If we wish to examine pipelining as well, we may consider extending these flows with line workflows (we need to tune the distribution of blocking and non-blocking operations in these flows too). In addition, to further enrich our workflows, we may also consider having multiple "bodies" in our design, which can represent not necessarily data warehouse tables, but ETL activities as well.

Moreover, having in hand such categorization one may decompose existing complex ETL workflows into sets of primitive constructs for getting insight into their functionality. This decomposition can be used for optimization purposes too. We can study the behavior of the abovementioned ETL patterns in isolation, and then, we can use our findings for optimizing and tuning the whole workflow for performance, maintainability or some other quality. For example, the performance of a complex workflow can be derived from the performance of the component primitive ones.

2.4 Goals of the Benchmark

The design of a benchmark should be based upon a clear understanding of the characteristics of the inspected systems that do matter. Therefore, we propose a configuration that covers a broad range of possible workflows (i.e., a large set of configurable parameters) and a limited set of monitored measures.

The goal of this benchmark is to provide the experimental testbed to be used for the assessment of ETL engines and design methods concerning their basic behavioral properties (measures) over a broad range of ETL workflows.

This benchmark's goal is to study and evaluate workflows as a whole. Here, we are not interested in providing *specialized performance measures for very specific tasks* in the overall process. We are not interested either, in exhaustively enumerating *all the possible alternatives for specific operations*. For example, this benchmark is not intended to facilitate the comparison of alternative methods for duplicate detection in a data set, since it does not take the tuning of all the possible parameters for this task under consideration. On the contrary, this benchmark can be used for the assessment of the integration of such methods in complex ETL workflows, assuming that all the necessary knobs have been appropriately tuned.

There are two modes of operation for ETL workflows: off-line (batch) and active (or continuous or real-time) modes. In the *off-line mode*, the workflow is executed during a specific time window (typically at night), when the systems are not servicing their end-users. Due to the low load of both the source and warehouse systems, the refreshment of data and any other administrative activities (cleanups, auditing, and so on) are easier to complete. In the *active mode*, the sources continuously try to send new data to the warehouse. This is not necessarily done instantly; rather, small groups of data are collected and sent to the warehouse for further processing. The two modes do not differ only on the frequency of the workflow execution, but also on how the workflow execution affects the load of the systems too.

Independently of the mode under which the ETL workflow operates, the two fundamental goals that should be reached are *effectiveness* and *efficiency*. Hence, given

an ETL engine or a specific design method to be assessed over one or more ETL workflows, these fundamental goals should be evaluated.

Effectiveness. Our extensive discussions with ETL practitioners and experts have verified that in real-life ETL projects performance is not the only objective. On the contrary, other optimization qualities are of interest as well. We refer to these collectively as QoX [6]. The QoX metric suite is incorporated at all stages of the design process, from high-level specifications to implementation. A non-exhaustive list of metrics that can be used to guide optimization include: performance, recoverability, reliability, freshness, maintainability, scalability, availability, flexibility, robustness, affordability, consistency, traceability, and auditability. Some metrics are quantitative (e.g., reliability, freshness, cost) while other metrics may be difficult to quantify (e.g., maintainability, flexibility). Also, there are significant tradeoffs that should be taken under consideration, since an effort for improving one objective may hurt another one [13]. For example, improving freshness typically hurts recoverability, since considering recovery points on the way to the warehouse may be prohibitive in this case; on the other hand, having redundancy may be an interesting solution for achieving fault-tolerance. Due to space consideration, we do not elaborate on all the abovementioned measures (for a more detailed discussion we refer to [13]).

However, the main objective is to have data respect both database and business rules. We believe that the following (non-exhaustive) list of questions should be considered in the creation of an ETL benchmark:

Q1. Does the workflow execution reach the maximum possible level of data *freshness*, *completeness*, and *consistency* in the warehouse within the necessary time (or resource) constraints?

Q2. Is the workflow execution *resilient to* occasional *failures*?

Q3. Is the workflow easily *maintainable*?

Freshness. A clear business rule is the need to have data as fresh as possible in the warehouse. Also, we need all of the source data to be eventually loaded at the warehouse; the update latency depends on the freshness requirements. Nevertheless, the sources and the warehouse must be consistent at least at a certain frequency (e.g., at the end of a day).

Missing changes at the source. Depending on what kind of change detector we have at the source, it is possible that some changes are lost (e.g., if we have a log sniffer, bulk updates not passing from the log file are lost). Also, in an active warehouse, if the active ETL engine needs to shed some incoming data in order to be able to process the rest of the incoming data stream successfully, it is imperative that these left-over tuples need to be processed later.

Recovery from failures. If some data are lost from the ETL process due to failures, then, we need to synchronize sources and warehouse and compensate the missing data. Of course, tuples from aborted transactions that have been sent to the warehouse (or they are on their way to it) should be undone.

Maintainability. In addition, keeping the ETL workflow maintainable is crucial for the cost of ETL lifecycle. A number of parameters may affect the maintainability of the system. Here, we focus on parameters indicating the cost of handling evolution events during the ETL lifecycle. Ideally, a simple ETL design is more maintainable, whereas in a complex one it is more difficult to keep track of a change.

Efficiency. Efficiency is an important aspect of ETL design. Since typically ETL processes should run within strict time windows, performance does matter. In fact, achieving high performance is not only important per se, it can also serve as a means for enabling (or achieving) other qualities as well. For example, a typical technique for achieving recoverability is to add recovery points to the ETL workflow. However, this technique is time-consuming (usually, the i/o cost of maintaining recovery points is significant), so in order to meet the execution time requirements, we need to boost ETL performance. Typical questions need to be answered are as follows:

Q4. How *fast* is the workflow executed?

Q5. What degree of *parallelization* is required?

Q6. How much *pipelining* does the workflow use?

Q7. What *resource overheads* does the workflow incur at the source, intermediate (staging), and warehouse sites?

Parallelization. The configuration in terms of parallelism plays an important role for the performance of an ETL process. In general, there exist two broad categories of parallel processing: pipelining and partitioning. In pipeline parallelism, the various activities are operating simultaneously in a system with more than one processor. This scenario performs well for ETL processes that handle a relative small volume of data. For large volumes of data, a different parallelism policy should be devised: the partitioning of the dataset into smaller sets. Then, we use different instances of the ETL process for handling each partition of data. In other words, the same activity of an ETL process would run simultaneously by several processors, each processing a different partition of data. At the end of the process, the data partitions should be merged and loaded to the target recordset(s). Frequently, a combination of the two policies is used to achieve maximum performance. Hence, while an activity is processing partitions of data and feeding pipelines, a subsequent activity may start operating on a certain partition before the previous activity had finished.

Minimal overheads at the sources and the warehouse. The production systems are under continuous load due to the large number of OLTP transactions performed simultaneously. The warehouse system supports a large number of readers executing client applications or decision support queries. In the offline ETL, the overheads incurred are of rather secondary importance, since the contention with such processes is practically non-existent. Still, in active warehousing, the contention is clear.

- *Minimal overhead of the source systems*. It is imperative to impose the minimum additional workload to the source, in the presence of OLTP transactions.
- *Minimal overhead of the DW system*. As the warehouse is populated by loading processes, other processes ask data from it. Then, the desideratum is that the warehouse operates with the lightest possible footprints for the loading processes as well as the minimum possible delay for incoming tuples and user queries.

3. Benchmark Parameters

In this section, we propose a set of configuration parameters along with a set of measures to be monitored in order to assess the fulfillment of the benchmark goals.

Experimental parameters. The following *problem parameter*s are of particular importance to the measurement of ETL workflows:

P1. the *size of the workflow* (i.e., the number of nodes contained in the graph),

P2. the *structure of the workflow* (i.e., the variation of the nature of the involved nodes and their interconnection as the workflow graph),

P3. the *size of input data* originating from the sources,

P4. the *workflow selectivity*, based on the selectivities of the workflow activities,

P5. the values of *probabilities of failure*,

P6. the *latency* of updates at the warehouse (i.e., it captures freshness requirements),

P7. the required *completion time* (i.e., this reflects the maximum tolerated execution time window),

P8. the system *resources* (e.g., memory and processing power), and

P9. the "*ETL workload*" that determines an execution *order* for ETL workflows and the *number* of instances of the workflows that should run concurrently (e.g., for evaluating parallelization in an ETL engine, one may want to run first a complex ETL workload composed of a high number of line workflows that should run in parallel, and then, a smaller set of tree workflows for merging the former ones).

Measured Effects. For each set of experimental measurement, certain *measures* need to be assessed, in order to characterize the fulfillment of the aforementioned goals. In the sequel, we classify these measures according to the assessment question they are employed to answer.

Q1. Measures for data freshness and data consistency. The objective is to have data respect both database and business rules. Also, we need data to be consistent with respect to the source as much as possible. The latter possibly incurs a certain time window for achieving this goal (e.g., once a day), in order to accommodate high refresh rates in the case of active data warehouses or failures in the general case. Concrete measures are:

- (M1.1) Percentage of data that violate business rules.
- (M1.2) Percentage of data that should be present at their appropriate warehouse targets, but they are not.

Q2. Measures for the resilience to failures. The main idea is to perform a set of workflow executions that are intentionally abnormally interrupted at different stages of their execution. The objective is to discover how many of these workflows were successfully compensated within the specified time constraints. For achieving resilience to failures, we consider two strategies or quality objectives: recoverability and redundancy. For the former, the most typical technique is to enrich the ETL process with recovery points (used for intermediate staging of data processed up to that point), so that after a failure the process may resume from the latest recovery point. However, where to put such points is not a straightforward task. Redundancy can be achieved with three techniques: replication, diversity or fail-over. For lack of space, here we refer only to replication, which involves multiple instances of the same process (or of a part of it) that run in parallel. Concrete measures are:

- (M2.1) Percentage of successfully resumed workflow executions.
- (M2.2) MTBF, the mean time between failures.

- (M2.3) MTTR, mean time to repair.
- (M2.4) Number of recovery points used.
- (M2.5) Resumption type: synchronous or asynchronous.
- (M2.6) Number of replicated processes (for replication).
- (M2.7) Uptime of ETL process.

Q3. Measures for maintainability. Maintainability is a qualitative objective and finding measures to evaluate it is more difficult than the other quantitative objectives (e.g, performance or recoverability). An approach to this, is to consider the effort for modifying the process after a change has been occurred either at the SLA's (service level agreements) or the underlying systems (e.g., after adding, renaming or deleting an attribute or a table at a source site). Concrete measures are:

- (M3.1) Length of the workflow or in other words, the length of its longest path (i.e., how far in the process a change should be propagated).
- (M3.2) Complexity of the workflow refers to the amount of relationships that combine its components [3].
- (M3.3) Modularity (or cohesion) refers to the extent to which the workflow components perform exactly one job; thus, a workflow is more modular if it contains less sharable components. Modularity imposes some interesting tradeoffs, for example with parallelization.
- (M3.4) Coupling captures the amount of relationship among different recordsets or activities (i.e., workflow components).

Q4. Measures for the speed of the overall process. The objective is to perform the ETL process as fast as possible. In the case of off-line loading, the objective is to complete the process within the specified time-window. Naturally, the faster this is performed the better (especially, in the context of failure resumption). In the case of active warehouse, where the ETL process is performed very frequently, the objective is to minimize the time that each tuple spends inside the ETL module. Concrete measures are:

- (M4.1) Throughput of regular workflow execution (this may also be measured as total completion time).
- (M4.2) Throughput of workflow execution including a specific percentage of failures and their resumption.
- (M4.3) Average latency per tuple in regular execution.

Q5. Measures for partitioning. The partitioning parallelism is affected by a set of choices. Partitioning a flow is not straightforward, since the splitting and especially, the merging operations required for the partitioning do not come without a cost. Concrete measures are:

- (M5.1) Partition type (e.g., round-robin, hash-based, follow-database-partitioning, and so on), which should be chosen according the characteristics of the workflow. For example, a flow heavy on sort-based operations may consider hash-based partitioning instead of round-robin.
- (M5.2) Number and length of workflow parts that use partitioning.
- (M5.3) Number of partitions.

– (M5.4) Data volume in each partition (this is related to partition type too).

Q6. Measures for pipelining. The pipelining parallelization is affected by parts of the workflow that contain (or not) blocking operations (e.g., transformations based on sort or aggregation). Concrete measures are:

– (M6.1) CPU and memory utilization for pipelining flows or for individual operation run in such flows.
– (M6.2) Min/Max/Avg length of the largest and smaller paths (or subgraphs) containing pipelining operations.
– (M6.3) Min/Max/Avg number of blocking operations.

Q7. Measured Overheads. The overheads at the source and the warehouse can be measured in terms of consumed memory and latency with respect to regular operation. Concrete measures are:

– (M7.1) Min/Max/Avg/ timeline of memory consumed by the ETL process at the source system.
– (M7.2) Time needed to complete the processing of a certain number of OLTP transactions in the presence (as opposed to the absence) of ETL software at the source, in regular source operation.
– (M7.3) The same as 7.2, but in the case of source failure, where ETL tasks are to be performed too, concerning the recovered data.
– (M7.4) Min/Max/Avg/ timeline of memory consumed by the ETL process at the warehouse system.
– (M7.5) (active warehousing) Time needed to complete the processing of a certain number of decision support queries in the presence (as opposed to the absence) of ETL software at the warehouse, in regular operation.
– (M7.6) The same as M7.5, but in the case of any (source or warehouse) failure, where ETL tasks are to be performed too at the warehouse side.

4. Specific Scenarios

A particular problem that arises in designing a test suite for ETL workflows concerns the complexity (structure and size) of the employed workflows. A means to deal with this is to construct a *workflow generator*, based on the aforementioned disciplines. Another means is to come up with an *indicative set of ETL workflows* that serve as the basis for experimentations. For space consideration, here we present the latter and we propose a small, exemplary set of specific ETL flows based on the TPC-H [16].

4.1 Database Schema

The information kept in the warehouse concerns parts and their suppliers as well as orders that customers have along with demographic data for the customers. The scenarios used in the experiments clean and transform the source data into the desired *warehouse schema*. The sources for our experiments are of two kinds, the *storage*

houses and *sales points*. Every storage house keeps data for the suppliers and parts, while every sales point keeps data for the customers and the orders. (The schemata of the sources and the data warehouse are depicted in Figure A1 – in the appendix.)

4.2 ETL Scenarios

We consider the butterfly cases discussed in Section 2 to be representative of a large number of ETL scenarios and thus, we propose a specific scenario for each kind. Due to space limitation, here we provide only small-size scenarios indicatively (e.g., a right-deep scenario is not given). However, as we discussed, one may create larger scenarios based on these exemplary structures. The scenarios are depicted in Figure 2 (their detailed descriptions can be found in the appendix of this paper).

The **line** workflow has a simple form since it applies a set of filters, transformations, and aggregations to a single table. This scenario type is used to filter source tables and assure that the data meet the logical constraints of the data warehouse.

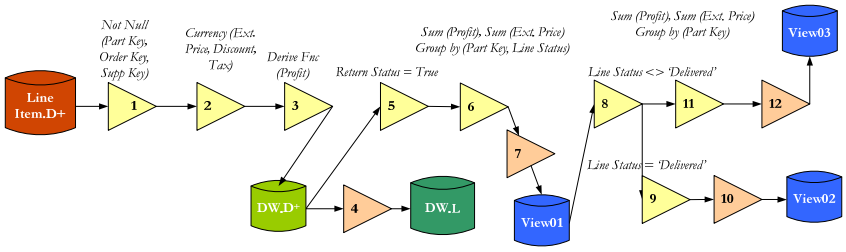
A **wishbone** workflow joins two parallel lines into one. This scenario is preferred when two tables in the source database should be joined in order to be loaded to the data warehouse or in the case where we perform similar operations to different data that are later joined. In our exemplary scenario, we track the changes that happen in a source containing customers. We compare the customers of the previous load to the ones of the current load and search for new customers to be loaded in the warehouse.

The **primary flow** scenario is a common scenario in cases where the source table must be enriched with surrogate keys. This exemplary primary flow that we use has as input the *Orders* table. The scenario is simple: all key-based values ("orderstatus", "custkey", "orderkey") pass through surrogate key filters that lookup (join) the incoming records in the appropriate lookup table. The resulting rows are appended to the relation *DW.Orders*. If incoming records exist in the *DW.Orders* relation and they have changed values then they are overwritten (thus, the Slowly Changing Dimension Type 1 tag in the figure); otherwise, a new entry is inserted in the warehouse relation.

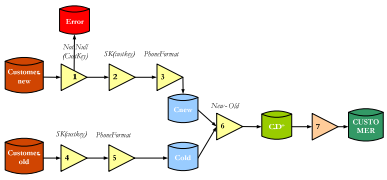
The **tree** scenario joins several source tables and applies aggregations on the result recordset. The join can be performed over either heterogeneous relations, whose contents are combined, either over homogeneous relations, whose contents are integrated into one unified (possible sorted) data set. In our case, the exemplary scenario involves three sources for the warehouse relation *PartSupp*.

The **fork** scenario applies a set of aggregations on a single source table. First the source table is cleaned, just like in a line scenario and the result table is used to create a set of materialized views. Our exemplary scenario uses the *Lineitem* table as the butterfly's body and starts with a set of extracted new records to be loaded.

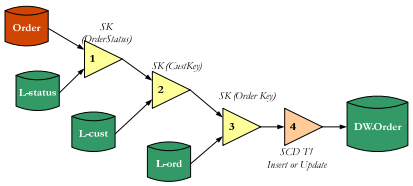
The most general-purpose scenario type is a **butterfly** scenario. It joins two or more source tables before a set of aggregations is performed on the result of the join. The left wing of the butterfly joins the source tables, while the right wing performs the desired aggregations producing materialized views. Our first exemplary scenario uses new source records concerning *Partsupp* and *Supplier* as its input. A second exemplary scenario introduces a Slowly Changing Dimension plan, populating the dimension table *PART* and retaining its history at the same time.



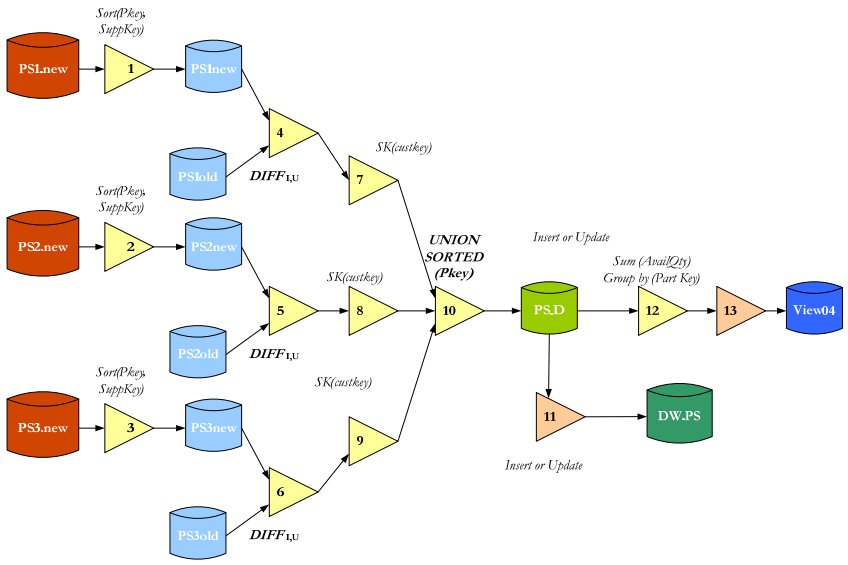
Line



Wishbone

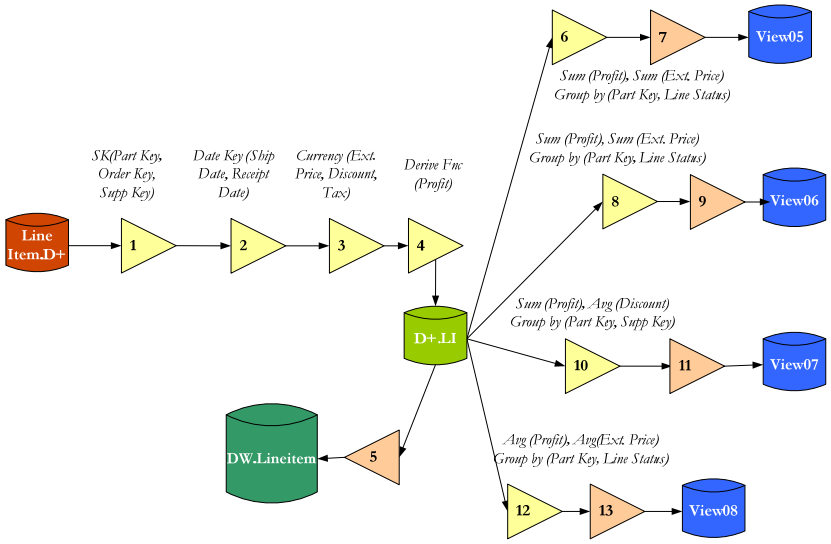


Primary Flow

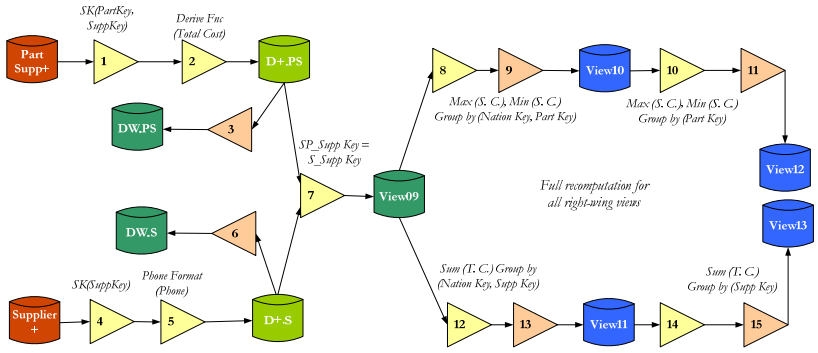


Tree

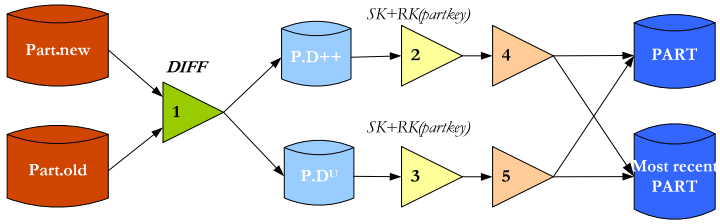
Figure 2. Specific ETL workflows



Fork



Balanced Butterfly (1)



Balanced Butterfly (2)

Figure 2. Specific ETL workflows (continued)

5. Related Work

Several benchmarks have been proposed in the database literature, in the past. Most of the benchmarks that we have reviewed make careful choices: (a) on the database schema & instance they use, (b) on the type of operations employed and (c) on the measures to be reported. Each benchmark has a guiding goal, and these three parts of the benchmark are employed to implement it.

As an example, we mention two benchmarks mainly coming from the Wisconsin database group. The OO7 benchmark was one of the first attempts to provide a comparative platform for object-oriented DBMS's [4]. The OO7 benchmark had the clear target to test as many aspects as possible of the efficiency of the measured systems (speed of pointer traversal, update efficiency, query efficiency). The BUCKY benchmark had a different viewpoint: the goal was to narrow down the focus only on the aspects of an OODBMS that were object-oriented (or object-relational): queries over inheritance, set-valued attributes, pointer navigation, methods and ADTS [5]. Aspects covered by relational benchmarks were not included in the BUCKY benchmark.

TPC has proposed two benchmarks for the case of decision support. The TPC-H benchmark [16] 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 consists of eight separate tables with 5 of them being clearly dimension tables, one being a clear fact table and a couple of them combinations of fact and dimension tables. *Unfortunately, the refreshment operations provided by the benchmark are primitive and not particularly useful as templates for the evaluation of ETL scenarios*.

TPC-DS is a new Decision Support (DS) workload being developed by the TPC [11, 15]. 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. There are three sales channels: store, catalog and the web. There are two fact tables in each channel, sales and returns, and a total of seven fact tables. In this dataset, the row counts for tables scale differently per table category: specifically, in fact tables the row count grows linearly, while in dimension tables grows sub-linearly. This benchmark also provides refreshment scenarios for the data warehouse. Still, *all these scenarios belong to the category of primary flows*, in which surrogate and global keys are assigned to all tuples. Recently, a new effort has been started driven by the TPC-ETL committee, but so far, concrete results have not been reported [15].

An early version of this paper was presented in [17]; due to lack of formal proceedings, please refer to the online version.

6. Conclusions

In this paper, we have dealt with the challenge of presenting a unified experimental playground for ETL processes. First, we have presented a principled way for constructing ETL workflows and we have identified their most prominent elements. We

have classified the most frequent ETL operations based on their special characteristics. We have shown that this classification adheres to the built-in operations of three popular commercial ETL tools; we do not anticipate any major deviations for other tools. Moreover, we have proposed a generic categorization of ETL workflows, namely butterflies, which covers frequent design cases. We have identified the main parameters and measures that are crucial in ETL environment and we have discussed how parallelism affects the execution of an ETL process. Finally, we have proposed specific ETL scenarios based on the aforementioned analysis, which can be used as an experimental testbed for the evaluation of ETL methods or tools.

Open issues involve (a) the handling of non-relational data, the treatment of near real time ETL, (c) the tuning of several parameters of the benchmark with values that reflect real-world applications, (d) the handling of indexes, materialized views and auxiliary data structures at the target side of the warehouse, and (e) the treatment of platform and hardware characteristics. Extra care should be taken also for the control flow part of ETL processes.

The main message from our work is the need for a commonly agreed benchmark that reflects real-world ETL scenarios, both for research purposes and, ultimately, for the comparison of ETL tools. Feedback is necessary for further tuning the benchmark.

References

[1] Ab Initio. In the Web, available at: http://www.abinitio.com/, 2009.

[2] J. Adzic, V. Fiore. Data Warehouse Population Platform. In DMDW, 2003.

[3] L.C. Briand, S. Morasca, V.R. Basili. Property-Based Software Engineering Measurement. In IEEE Trans. on Software Engineering, 22(1), 1996.

[4] M. J. Carey, D. J. DeWitt, J. F. Naughton. The OO7 Benchmark. In SIGMOD, 1993.

[5] M. J. Carey et al. The BUCKY Object-Relational Benchmark. In SIGMOD, 1997.

[6] U. Dayal, M. Castellanos, A. Simitsis, K. Wilkinson. Data Integration Flows for Business Intelligence. In EDBT, 2009.

[7] IBM, " IBM InfoSphere Information Server", in the Web, available at: http://www-01.ibm.com/software/data/integration/info_server_platform/, 2009.

[8] Informatica, "PowerCenter", in the Web, available at: http://www.informatica.com/products/powercenter/, 2009.

[9] Microsoft. SQL Server Integration Services (SSIS), in the Web, available at: http://technet.microsoft.com/en-us/sqlserver/bb331782.aspx, 2009

[10] Oracle, "Oracle Warehouse Builder 11g", in the Web, available at: http://www.oracle.com/technology/products/warehouse/, 2009.

[11] R. Othayoth, M. Poess. The Making of TPC-DS. In VLDB, 2006.

[12] A. Simitsis, P. Vassiliadis, S. Skiadopoulos, T. Sellis. Data Warehouse Refreshment. In "Data Warehouses and OLAP: Concepts, Architectures and Solutions", IRM Press, 2006.

[13] A. Simitsis, K. Wilkinson, M. Castellanos, U. Dayal. QoX-Driven ETL Design: Reducing the Cost of the ETL Consulting Engagements. In SIGMOD, 2009.

[15] TPC. TPC Benchmark Status. TPC-ETL, in the Web, available at: http://www.tpc.org/reports/status/, 2009.

[16] TPC. TPC-H benchmark. Transaction Processing Council, in the Web, available at: http://www.tpc.org/, 2009.

[17] P. Vassiliadis, A. Karagiannis, V. Tziovara, A. Simitsis. Towards a Benchmark for ETL Workflows. In QDB, 2007, in the Web, available at: http://www.cs.uoi.gr/~pvassil/publications/publications.html

Appendix

The schemata of the sources and the data warehouse are depicted in Figure A1.

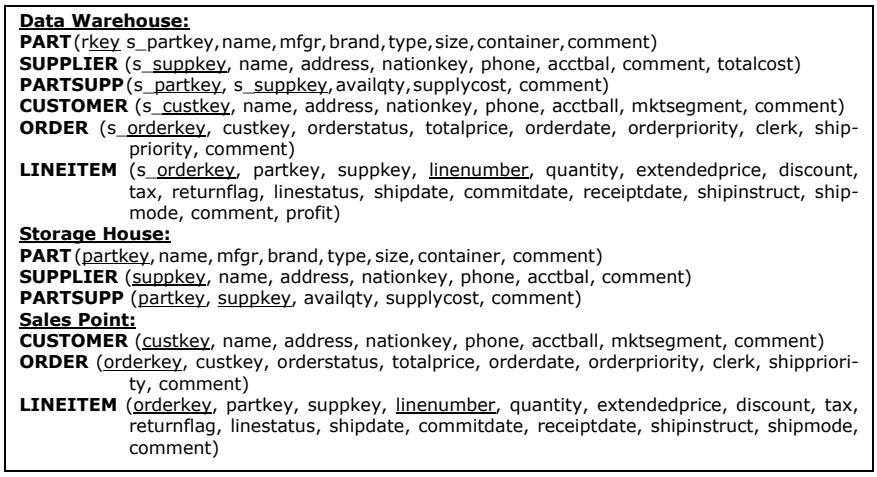


Figure A1. Database schemata

Detailed description of scenarios

Line. In the proposed scenario, we start with an extracted set of new source rows *LineItem.D+* and push them towards the warehouse as follows:

1. First, we check the fields "partkey", "orderkey" and "suppkey" for NULL values. Any NULL values are replaced by appropriate special values.
2. Next, a calculation of a value "profit" takes place. This value is locally derived from other fields in a tuple as the amount of "extendedprice" subtracted by the values of the "tax" and "discount" fields.
3. The third activity changes the fields "extendedprice", "tax", "discount" and "profit" to a different currency.
4. The results of this operation are loaded first into a delta table *DW.D+* and subsequently into the data warehouse *DWH*. The first load simply replaces the respective recordset, whereas the second involves the incremental appending of these rows to the warehouse.
5. The workflow is not stopped after the completion of the left wing, since we would like to create some materialized views. The next operation is a filter that keeps only records whose return status is "False".
6. Next, an aggregation calculates the sum of "extendedprice" and "profit" fields grouped by "partkey" and "linestatus".
7. The results of the aggregation are loaded in view *View01* by (a) updating existing rows and (b) inserting new groups wherever appropriate.

8. The next activity is a router, sending the rows of view *View01* to one of its two outputs, depending on the "linestatus" field has the value "delivered" or not.
9. The rows with value “delivered” are further aggregated for the sum of "profit" and "extendedprice" fields grouped by "partkey".
10. The results are loaded in view *View02* as in the case for view *View01*.
11. The rows with value different than “delivered” are further aggregated for the sum of "profit" and "extendedprice" fields grouped by "partkey".

    The results are loaded in view *View03* as in the case for view *View01*.

Wishbone. The scenario evolves as follows:

1. The first activity on the new data set checks for NULL values in the "custkey" field. The problematic rows are kept in an error log file for further off-line processing.
2. Both previous and old data are passed through a surrogate key transformation. We assume a domain size that fits in main memory for this source; therefore, the transformation is not performed as a join with a lookup table, but rather as a lookup function call invoked per row.
3. Moreover, the next activity converts the phone numbers in a numeric format, removing dashes and replacing the '+' character with the "00" equivalent.
4. The transformed recordsets are persistently stored in relational tables or files which are subsequently compared through a difference operator (typically implemented as a join variant) to detect new rows.
5. The new rows are stored in a file $C.D^{+}$ which is kept for the possibility of failure. Then the rows are appended in the warehouse dimension table *Customer*.

Tree. The scenario evolves as follows:

1. Each new version of the source is sorted by its primary key and checked against its past version for the detection of new or updated records. The $DIFF_{I,U}$ operator checks the two inputs for the combination of pkey, suppkey matches. If a match is not found, then a new record is found. If a match is found and there is a difference in the field “availqty” then an update needs to be performed.
2. These new records are assigned surrogate keys per source
3. The three streams of tuples are united in one flow and they are also sorted by “pkey” since this ordering will be later exploited. Then, a delta file *PS.D* is produced.
4. The contents of the delta file are appended in the warehouse relation *DW.PS*.

At the same time, the materialized view *View04* is refreshed too. The delta rows are summarized for the available quantity per pkey and then, the appropriate rows in the view are either updated (if the group exists) or (inserted if the group is not present).

Fork. The fork scenario evolves as follows:

1. Surrogate keys are assigned to the fields "partkey", "orderkey" and "suppkey".
2. We convert the dates in the "shipdate" and "receiptdate" fields into a “dateId”, a unique identifier for every date.
3. The third activity is a calculation of a value "profit". This value is derived from other fields in every tuple as the amount of "extendedprice" subtracted by the values of the "tax" and "discount" fields.
4. This activity changes the fields "extendedprice", "tax", "discount" and "profit" to a different currency. The result of this actvity is stored at a delta table $D^{+}.LI$. The records are appended to the data warehouse *LineItem* table and they are also reused for a number of aggregations at the right wing of the butterfly. All records pushed towards the views, either update or insert new records in the views, depending on the existence (or not) of the respective groups.
5. The aggregator for *View05* calculates the sum of the "profit" and "extendedprice" fields grouped by the "partkey" and "linestatus" fields.
6. The aggregator for *View06* calculates the sum of the "profit" and "extendedprice" fields grouped by the "linestatus" fields.
7. The aggregator for *View07* calculates the sum of the "profit" field and the average of the "discount" field grouped by the "partkey" and "suppkey" fields.
8. The aggregator for *View08* calculates the average of the "profit" and "extendedprice" fields grouped by the "partkey" and "linestatus" fields.

Butterfly. The first scenario uses *Partsupp* and *Supplier* as its input.

1. Concerning the *Partsupp* source, we generate surrogate key values for the "partkey" and "suppkey" fields. Then, the "totalcost" field is calculated and added to each tuple.
2. Then, the transformed records are saved in a delta file *D+.PS* and appended to the relation *DW.Partsupp*.
3. Concerning the *Supplier* source, a surrogate key is generated for the “suppkey” field and a second activity transforms the "phone" field.
4. Then, the transformed records are saved in a delta file *D+.S* and appended to the relation *DW.Supplier*.
5. The delta relations are subsequently joined on the "ps_suppkey" and "s_suppkey" fields and populate the view *View09*, which is augmented with the new records. Then, several views are computed from scratch, as follows.
6. View *View10* calculates the maximum and the minimum value of the "supplycost" field grouped by the "nationkey" and "partkey" fields.
7. *View12* calculates the maximum and the minimum of the "supplycost" field grouped by the "partkey" fields.
8. *View11* calculates the sum of the "totalcost" field grouped by the "nationkey" and "suppkey" fields.
9. *View13* calculates the sum of the "totalcost" field grouped by the "suppkey" field.

The second butterfly scenario concerns Slowly Changing dimensions, populating the dimension table *PART* and retaining its history at the same time. The trick is found in the combination of the "rkey", "s_partkey" attributes. The "s_partkey" assigns a surrogate key to a certain tuple (e.g., assume it assigns 10 to a product *X*). If the product changes in one or more attributes at the source (e.g., X's "size" changes), then a new record is generated, with the same "s_partkey" and a different "rkey" (which can be a timestamp-based key, or similar). The scenario works as follows:

1. A new and an old version of the source table *Part* are compared for changes. Changes are directed to *P.D++* (for new records) and *P.DU* for updates in the fields "size" and "container"
2. Surrogate and recent keys are assigned to the new records that are propagated to the table *PART* for storage.
3. An auxiliary table *MostRecentPART* holding the most recent "rkey" per "s_partkey" is appropriately updated.

Observe that in this scenario the body of the butterfly is an activity.

Statistics

Figure A2 presents summarized statistics of the constituents of the ETL workflows depicted in Figure 2. Such statistics reveal the functionality (i.e., the nature) of each workflow. (The numbers L+R refer to the left (L) and right (R) wings, respectively.)

	Filters	**Functions**	**Routers**	**Aggr**	**Holistic f.**	**Joins**	**Diff**	**Unions**	**Load Body**	**Load Views**
Line	1+1	2+0	0+1	0+3					INCR	INCR
Wishbone	1+0	4+0				1+0			INCR	-
Pr. Flow						3+0			I/U	-
Tree				0+1	1+0	1+0		1+0	I/U	I/U
Fork		3+0		0+4					INCR	INCR
BB(1)		4+0		0+4		1+0			INCR	FULL
BB(2)		0+2					1		-	I/U
	2+1	13+2	0+1	0+12	1+0	6+0	1	1+0		

Figure A2. Statistics of the proposed ETL workflows

Taxonomy of activities

Figure A3 presents a taxonomy of activities at the micro level and similar built-in transformations provided by commercial ETL tools. For each category of activities presented in Section 2.2, a representative set of transformations, which are provided by three popular commercial ETL tools, is presented. The figure is indicative and in many ways incomplete. The goal is not to provide a comparison among the three tools. On the contrary, we would like to stress out the genericity of our classification. For most ETL tools, the set of built-in transformations is enriched by user defined operations and a plethora of functions. Still, as figure A3 shows, all frequently built-in transformations existing in commercial solutions fall into our classification.

	Transformation Category*	SQL Server Information Services SSIS	DataStage	Oracle Warehouse Builder
Transformation and Cleansing	**Row-level**: Function that can be applied locally to a single row	– Character Map – Copy Column – Data Conversion – Derived Column – Script Component – OLE DB Command – Other filters (not null, selections, etc.)	– Transformer (A generic representative of a broad range of functions: date and time, logical, mathematical, null handling, number, raw, string, utility, type conversion/casting, routing.) – Remove duplicates – Modify (drop/keeps columns or change their types)	– Deduplicator (distinct) – Filter – Sequence – Constant – Table function (it is applied on a set of rows for increasing the performance) – Data Cleaning Operators (Name and Address, Match-Merge) – Other SQL transformations (Character, Date, Number, XML)
	Routers: Locally decide, for each row, which of the many outputs it should be sent to	– Conditional Split – Multicast	– Copy – Filter – Switch	– Splitter
	Unary Grouper: Transform a set of rows to a single row	– Aggregate – Pivot/Unpivot	– Aggregator – Make/Split subrecord – Combine/Promote records – Make/Split vector	– Aggregator – Pivot/Unpivot
	Unary Holistic: Perform a transformation to the entire data set (blocking)	– Sort – Percentage Sampling – Row Sampling	– Sort (sequential, parallel, total)	– Sorter
	Binary or N-ary: Combine many inputs into one output	Union-like: – Union All – Merge Join-like: – Merge Join (MJ) – Lookup (SKJ) – Import Column (NLJ)	Union-like: – Funnel (continuous, sort, sequence) Join-like: – Join – Merge – Lookup Diff-like: – Change capture/apply – Difference (record-by-record) – Compare (column-by-column)	Union-like: – Set (union, union all, intersect, minus) Join-like: – Joiner – Key Lookup (SKJ)
Extr.		– Import Column Transformation	– Compress/Expand – Column import	– Merge – Import
Load		– Export Column – Slowly Changing Dimension	– Compress/Expand – Column import/export	– Merge – Export – Slowly Changing Dimension

* All ETL tools provide a set of physical operations that facilitate either the extraction or the loading phase. Such operations include: extraction from hashed/sequential files, delimited/fixed width/multi-format flat files, file set, ftp, lookup, external sort, compress/uncompress, and so on.

Figure A3. Taxonomy of ETL activities