Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Čeština
Deutsch
Español
Français
한국어
Italiano
עברית
Latviešu
Lumbaart
Nederlands
日本語
Norsk
Polski
Português
Русский
Українська
中文
Edit links

Extract, transform, load

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help **improve it** or discuss these issues on the **talk page**. *(Learn how and when to remove these template messages)*

- This article includes a list of references, but **its sources remain unclear** because it has **insufficient inline citations**. *(November 2011)*
- This article **possibly contains original research**. *(December 2011)*

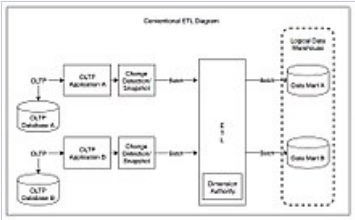
In computing, **extract, transform, load** (**ETL**) refers to a process in database usage and especially in data warehousing. The ETL process became a popular concept in the 1970s.[1] Data extraction is where data is extracted from homogeneous or heterogeneous data sources; data transformation is where the data is transformed for storing in the proper format or structure for the purposes of querying and analysis; data loading where the data is loaded into the final target database, more specifically, an operational data store, data mart, or data warehouse. A properly designed ETL system extracts data from the source systems, enforces data quality and consistency standards, conforms data so that separate sources can be used together, and finally delivers data in a presentation-ready format so that application developers can build applications and end users can make decisions.[2]

Since the data extraction takes time, it is common to execute the three phases in parallel. While the data is being extracted, another transformation process executes while processing the data already received and prepares it for loading while the data loading begins without waiting for the completion of the previous phases.

ETL systems commonly integrate data from multiple applications (systems), typically developed and supported by different vendors or hosted on separate computer hardware. The separate systems containing the original data are frequently managed and operated by different employees. For example, a cost accounting system may combine data from payroll, sales, and purchasing.

Contents

1 Extract
2 Transform
3 Load
4 Real-life ETL cycle
5 Challenges
6 Performance
7 Parallel processing
8 Rerunnability, recoverability
9 Virtual ETL
10 Dealing with keys
11 Tools
12 ETL vs. ELT
13 See also
14 References



Conventional ETL Diagram[3]

Extract [edit]

The first part of an ETL process involves extracting the data from the source system(s). In many cases, this represents the most important aspect of ETL, since extracting data correctly sets the stage for the success of subsequent processes. Most data-warehousing projects combine data from different source systems. Each separate system may also use a different data organization and/or format. Common data-source formats include relational databases, XML, JSON and flat files, but may also include non-relational

database structures such as Information Management System (IMS) or other data structures such as Virtual Storage Access Method (VSAM) or Indexed Sequential Access Method (ISAM), or even formats fetched from outside sources by means such as web spidering or screen-scraping. The streaming of the extracted data source and loading on-the-fly to the destination database is another way of performing ETL when no intermediate data storage is required. In general, the extraction phase aims to convert the data into a single format appropriate for transformation processing.

An intrinsic part of the extraction involves data validation to confirm whether the data pulled from the sources has the correct/expected values in a given domain (such as a pattern/default or list of values). If the data fails the validation rules it is rejected entirely or in part. The rejected data is ideally reported back to the source system for further analysis to identify and to rectify the incorrect records. In some cases, the extraction process itself may have to do a data-validation rule in order to accept the data and flow to the next phase.

Transform [edit]

In the data transformation stage, a series of rules or functions are applied to the extracted data in order to prepare it for loading into the end target. Some data does not require any transformation at all; such data is known as "direct move" or "pass through" data.

An important function of transformation is the cleaning of data, which aims to pass only "proper" data to the target. The challenge when different systems interact is in the relevant systems' interfacing and communicating. Character sets that may be available in one system may not be so in others.

In other cases, one or more of the following transformation types may be required to meet the business and technical needs of the server or data warehouse:

- Selecting only certain columns to load: (or selecting null columns not to load). For example, if the source data has three columns (aka "attributes"), roll_no, age, and salary, then the selection may take only roll_no and salary. Or, the selection mechanism may ignore all those records where salary is not present (salary = null).
- Translating coded values: (*e.g.*, if the source system codes male as "1" and female as "2", but the warehouse codes male as "M" and female as "F")
- Encoding free-form values: (*e.g.*, mapping "Male" to "M")
- Deriving a new calculated value: (*e.g.*, sale_amount = qty * unit_price)
- Sorting or ordering the data based on a list of columns to improve search performance
- Joining data from multiple sources (*e.g.*, lookup, merge) and deduplicating the data
- Aggregating (for example, rollup — summarizing multiple rows of data — total sales for each store, and for each region, etc.)
- Generating surrogate-key values
- Transposing or pivoting (turning multiple columns into multiple rows or vice versa)
- Splitting a column into multiple columns (*e.g.*, converting a comma-separated list, specified as a string in one column, into individual values in different columns)
- Disaggregating repeating columns
- Looking up and validating the relevant data from tables or referential files
- Applying any form of data validation; failed validation may result in a full rejection of the data, partial rejection, or no rejection at all, and thus none, some, or all of the data is handed over to the next step depending on the rule design and exception handling; many of the above transformations may result in exceptions, e.g., when a code translation parses an unknown code in the extracted data

Load [edit]

The load phase loads the data into the end target, which may be a simple delimited flat file or a data warehouse. Depending on the requirements of the organization, this process varies widely. Some data warehouses may overwrite existing information with cumulative information; updating extracted data is frequently done on a daily, weekly, or monthly basis. Other data warehouses (or even other parts of the same data warehouse) may add new data in a historical form at regular intervals—for example, hourly. To understand this, consider a data warehouse that is required to maintain sales records of the last year. This data warehouse overwrites any data older than a year with newer data. However, the entry of data for any one year window is made in a historical manner. The timing and scope to replace or append are strategic design choices dependent on the time available and the business needs. More complex systems can maintain a history and audit trail of all changes to the data loaded in the data warehouse.

As the load phase interacts with a database, the constraints defined in the database schema — as well as in triggers activated upon data load — apply (for example, uniqueness, referential integrity, mandatory fields), which also contribute to the overall data quality performance of the ETL process.

- For example, a financial institution might have information on a customer in several departments and each department might have that customer's information listed in a different way. The membership department might list the customer by name, whereas the accounting department might list the customer by number. ETL can bundle all of these data elements and consolidate them into a uniform presentation, such as for storing in a database or data warehouse.
- Another way that companies use ETL is to move information to another application permanently. For instance, the new application might use another database vendor and most likely a very different database schema. ETL can be used to transform the data into a format suitable for the new application to use.
- An example would be an Expense and Cost Recovery System (ECRS) such as used by accountancies, consultancies, and legal firms. The data usually ends up in the time and billing system, although some businesses may also utilize the raw data for employee productivity reports to Human Resources (personnel dept.) or equipment usage reports to Facilities Management.

Real-life ETL cycle [edit]

The typical real-life ETL cycle consists of the following execution steps:

1. Cycle initiation
2. Build reference data
3. Extract (from sources)
4. Validate
5. Transform (clean, apply business rules, check for data integrity, create aggregates or disaggregates)
6. Stage (load into staging tables, if used)
7. Audit reports (for example, on compliance with business rules. Also, in case of failure, helps to diagnose/repair)
8. Publish (to target tables)
9. Archive

Challenges [edit]

ETL processes can involve considerable complexity, and significant operational problems can occur with improperly designed ETL systems.

The range of data values or data quality in an operational system may exceed the expectations of designers at the time validation and transformation rules are specified. Data profiling of a source during data analysis can identify the data conditions that must be managed by transform rules specifications, leading to an amendment of validation rules explicitly and implicitly implemented in the ETL process.

Data warehouses are typically assembled from a variety of data sources with different formats and purposes. As such, ETL is a key process to bring all the data together in a standard, homogeneous environment.

Design analysis should establish the scalability of an ETL system across the lifetime of its usage---including understanding the volumes of data that must be processed within service level agreements. The time available to extract from source systems may change, which may mean the same amount of data may have to be processed in less time. Some ETL systems have to scale to process terabytes of data to update data warehouses with tens of terabytes of data. Increasing volumes of data may require designs that can scale from daily batch to multiple-day micro batch to integration with message queues or real-time change-data-capture for continuous transformation and update.

Performance [edit]

ETL vendors benchmark their record-systems at multiple TB (terabytes) per hour (or ~1 GB per second) using powerful servers with multiple CPUs, multiple hard drives, multiple gigabit-network connections, and lots of memory.

In real life, the slowest part of an ETL process usually occurs in the database load phase. Databases may perform slowly because they have to take care of concurrency, integrity maintenance, and indices. Thus,

for better performance, it may make sense to employ:

- *Direct path extract* method or bulk unload whenever is possible (instead of querying the database) to reduce the load on source system while getting high speed extract
- Most of the transformation processing outside of the database
- Bulk load operations whenever possible

Still, even using bulk operations, database access is usually the bottleneck in the ETL process. Some common methods used to increase performance are:

- Partition tables (and indices): try to keep partitions similar in size (watch for `null` values that can skew the partitioning)
- Do all validation in the ETL layer before the load: disable integrity checking (`disable constraint` ...) in the target database tables during the load
- Disable triggers (`disable trigger` ...) in the target database tables during the load: simulate their effect as a separate step
- Generate IDs in the ETL layer (not in the database)
- Drop the indices (on a table or partition) before the load - and recreate them after the load (SQL: `drop index` ...; `create index` ...)
- Use parallel bulk load when possible — works well when the table is partitioned or there are no indices (Note: attempt to do parallel loads into the same table (partition) usually causes locks — if not on the data rows, then on indices)
- If a requirement exists to do insertions, updates, or deletions, find out which rows should be processed in which way in the ETL layer, and then process these three operations in the database separately; you often can do bulk load for inserts, but updates and deletes commonly go through an API (using SQL)

Whether to do certain operations in the database or outside may involve a trade-off. For example, removing duplicates using `distinct` may be slow in the database; thus, it makes sense to do it outside. On the other side, if using `distinct` significantly (x100) decreases the number of rows to be extracted, then it makes sense to remove duplications as early as possible in the database before unloading data.

A common source of problems in ETL is a big number of dependencies among ETL jobs. For example, job "B" cannot start while job "A" is not finished. One can usually achieve better performance by visualizing all processes on a graph, and trying to reduce the graph making maximum use of parallelism, and making "chains" of consecutive processing as short as possible. Again, partitioning of big tables and their indices can really help.

Another common issue occurs when the data are spread among several databases, and processing is done in those databases sequentially. Sometimes database replication may be involved as a method of copying data between databases - it can significantly slow down the whole process. The common solution is to reduce the processing graph to only three layers:

- Sources
- Central ETL layer
- Targets

This approach allows processing to take maximum advantage of parallelism. For example, if you need to load data into two databases, you can run the loads in parallel (instead of loading into the first - and then replicating into the second).

Sometimes processing must take place sequentially. For example, dimensional (reference) data are needed before one can get and validate the rows for main "fact" tables.

Parallel processing [edit]

A recent development in ETL software is the implementation of parallel processing. It has enabled a number of methods to improve overall performance of ETL when dealing with large volumes of data.

ETL applications implement three main types of parallelism:

- Data: By splitting a single sequential file into smaller data files to provide parallel access
- Pipeline: allowing the simultaneous running of several components on the same data stream, e.g. looking up a value on record 1 at the same time as adding two fields on record 2
- Component: The simultaneous running of multiple processes on different data streams in the same job, e.g. sorting one input file while removing duplicates on another file

All three types of parallelism usually operate combined in a single job.

An additional difficulty comes with making sure that the data being uploaded is relatively consistent. Because multiple source databases may have different update cycles (some may be updated every few minutes, while others may take days or weeks), an ETL system may be required to hold back certain data until all sources are synchronized. Likewise, where a warehouse may have to be reconciled to the contents in a source system or with the general ledger, establishing synchronization and reconciliation points becomes necessary.

Rerunnability, recoverability [edit]

Data warehousing procedures usually subdivide a big ETL process into smaller pieces running sequentially or in parallel. To keep track of data flows, it makes sense to tag each data row with "row_id", and tag each piece of the process with "run_id". In case of a failure, having these IDs help to roll back and rerun the failed piece.

Best practice also calls for *checkpoints*, which are states when certain phases of the process are completed. Once at a checkpoint, it is a good idea to write everything to disk, clean out some temporary files, log the state, etc.

Virtual ETL [edit]

As of 2010, data virtualization had begun to advance ETL processing. The application of data virtualization to ETL allowed solving the most common ETL tasks of data migration and application integration for multiple dispersed data sources. Virtual ETL operates with the abstracted representation of the objects or entities gathered from the variety of relational, semi-structured, and unstructured data sources. ETL tools can leverage object-oriented modeling and work with entities' representations persistently stored in a centrally located hub-and-spoke architecture. Such a collection that contains representations of the entities or objects gathered from the data sources for ETL processing is called a metadata repository and it can reside in memory[4] or be made persistent. By using a persistent metadata repository, ETL tools can transition from one-time projects to persistent middleware, performing data harmonization and data profiling consistently and in near-real time.

Dealing with keys [edit]

Unique keys play an important part in all relational databases, as they tie everything together. A unique key is a column that identifies a given entity, whereas a foreign key is a column in another table that refers to a primary key. Keys can comprise several columns, in which case they are composite keys. In many cases, the primary key is an auto-generated integer that has no meaning for the business entity being represented, but solely exists for the purpose of the relational database - commonly referred to as a surrogate key.

As there is usually more than one data source getting loaded into the warehouse, the keys are an important concern to be addressed. For example: customers might be represented in several data sources, with their Social Security Number as the primary key in one source, their phone number in another, and a surrogate in the third. Yet a data warehouse may require the consolidation of all the customer information into one dimension.

A recommended way to deal with the concern involves adding a warehouse surrogate key, which is used as a foreign key from the fact table.[5]

Usually, updates occur to a dimension's source data, which obviously must be reflected in the data warehouse.

If the primary key of the source data is required for reporting, the dimension already contains that piece of information for each row. If the source data uses a surrogate key, the warehouse must keep track of it even though it is never used in queries or reports; it is done by creating a lookup table that contains the warehouse surrogate key and the originating key.[6] This way, the dimension is not polluted with surrogates from various source systems, while the ability to update is preserved.

The lookup table is used in different ways depending on the nature of the source data. There are 5 types to consider;[7] three are included here:

Type 1

The dimension row is simply updated to match the current state of the source system; the warehouse does not capture history; the lookup table is used to identify the dimension row to update or overwrite

Type 2

A new dimension row is added with the new state of the source system; a new surrogate key is assigned; source key is no longer unique in the lookup table

Fully logged

A new dimension row is added with the new state of the source system, while the previous dimension row is updated to reflect it is no longer active and time of deactivation.

Tools [edit]

By using an established ETL framework, one may increase one's chances of ending up with better connectivity and scalability.[*citation needed*] A good ETL tool must be able to communicate with the many different relational databases and read the various file formats used throughout an organization. ETL tools have started to migrate into Enterprise Application Integration, or even Enterprise Service Bus, systems that now cover much more than just the extraction, transformation, and loading of data. Many ETL vendors now have data profiling, data quality, and metadata capabilities. A common use case for ETL tools include converting CSV files to formats readable by relational databases. A typical translation of millions of records is facilitated by ETL tools that enable users to input csv-like data feeds/files and import it into a database with as little code as possible.

ETL tools are typically used by a broad range of professionals - from students in computer science looking to quickly import large data sets to database architects in charge of company account management, ETL tools have become a convenient tool that can be relied on to get maximum performance. ETL tools in most cases contain a GUI that helps users conveniently transform data, using a visual data mapper, as opposed to writing large programs to parse files and modify data types.

While ETL tools have traditionally been for developers and IT staff, the new trend is to provide these capabilities to business users so they can themselves create connections and data integrations when needed, rather than going to the IT staff.[8] Gartner refers to these non-technical users as Citizen Integrators.[9]

ETL vs. ELT [edit]

There are two broad categories of ETL tools. First-generation ETL tools run on-premises, and connect to on-premises data warehouses that usually run on special-purpose high-end hardware in organizations' data centers. In that environment, it makes sense to do as much prep work as possible (i.e. transformation) prior to loading data, to avoid consuming expensive compute cycles that analysts need on the dedicated hardware.

Since 2013, however, cloud-based data warehouses like Amazon Redshift, Google BigQuery, and Snowflake Computing have been able to provide nearly infinitely scalable computing power. This lets businesses forgo preload transformations and replicate raw data into their data warehouses, where it can transform them as needed using SQL. With modern tools, ETL, in effect, becomes ELT. Many people still use "ETL" to refer to either kind of data pipeline.

See also [edit]

- Architecture patterns (EA reference architecture)
- Create, read, update and delete (CRUD)
- Data cleansing
- Data integration
- Data mart
- Data mediation
- Data migration
- Electronic Data Interchange (EDI)
- Enterprise architecture
- Expense and Cost Recovery System (ECRS)
- Hartmann pipeline
- Legal Electronic Data Exchange Standard (LEDES)
- Metadata discovery
- Online analytical processing
- Spatial ETL

References [edit]

1. ^ ETL What it is and why it matters
2. ^ Ralph., Kimball, (2004). *The data warehouse ETL toolkit : practical techniques for extracting, cleaning, conforming, and delivering data*. Caserta, Joe, 1965-. Indianapolis, IN: Wiley. ISBN 0764579231. OCLC 57301227.
3. ^ Ralph., Kimball, (2004). *The data warehouse ETL toolkit : practical techniques for extracting, cleaning, conforming, and delivering data*. Caserta, Joe, 1965-. Indianapolis, IN: Wiley. ISBN 0764579231. OCLC 57301227.
4. ^ Virtual ETL
5. ^ Kimball, The Data Warehouse Lifecycle Toolkit, p 332
6. ^ Golfarelli/Rizzi, Data Warehouse Design, p 291
7. ^ Golfarelli/Rizzi, Data Warehouse Design, p 291
8. ^ "The Inexorable Rise of Self Service Data Integration". Retrieved 31 January 2016.
9. ^ "Embrace the Citizen Integrator".

v · t · e

Data warehouse	
Creating the data warehouse	
Concepts	Database · Dimension · Dimensional modeling · Fact · OLAP · Star schema · Aggregate
Variants	Anchor Modeling · Column-oriented DBMS · Data vault modeling · HOLAP · MOLAP · ROLAP · Operational data store
Elements	Data dictionary/Metadata · Data mart · Sixth normal form · Surrogate key
Fact	Fact table · Early-arriving fact · Measure
Dimension	Dimension table · Degenerate · Slowly changing
Filling	**Extract-Transform-Load (ETL)** · Extract · Transform · Load
Using the data warehouse	
Concepts	Business intelligence · Dashboard · Data mining · Decision support system (DSS) · OLAP cube · Data warehouse automation
Languages	Data Mining Extensions (DMX) · MultiDimensional eXpressions (MDX) · XML for Analysis (XMLA)
Tools	Business intelligence software · Reporting software · Spreadsheet
Related	
People	Bill Inmon · Ralph Kimball
Products	Comparison of OLAP Servers · Data warehousing products and their producers

Categories: Extract, transform, load tools | Data warehousing