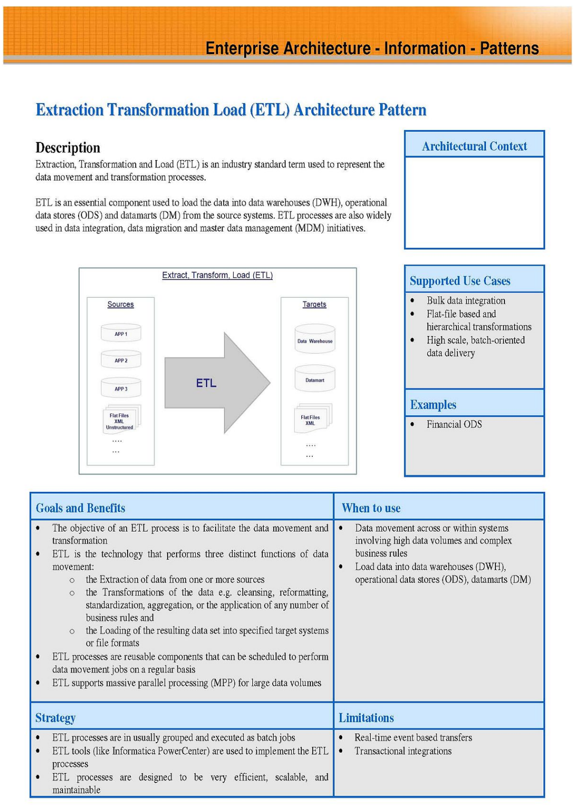
Extract, transform, load

In computing, **extract, transform and load** (**ETL**) refers to a process in database usage and especially in data warehousing that involves:

- Extracting data from outside sources
- Transforming it to fit operational needs (which can include quality levels)
- Loading it into the end target (database, more specifically, operational data store, data mart or data warehouse)

Extract

The first part of an ETL process involves extracting the data from the source systems. In many cases this is the most challenging aspect of ETL, as extracting data correctly will set the stage for how subsequent processes will go.



Enterprise Architecture - Information - Patterns

Extraction Transformation Load (ETL) Architecture Pattern

Description

Extraction, Transformation and Load (ETL) is an industry standard term used to represent the data movement and transformation processes.

ETL is an essential component used to load the data into data warehouses (DWH), operational data stores (ODS) and datamarts (DM) from the source systems. ETL processes are also widely used in data integration, data migration and master data management (MDM) initiatives.

Architectural Context

Supported Use Cases

- Bulk data integration
- Flat-file based and hierarchical transformations
- High scale, batch-oriented data delivery

Examples

- Financial ODS

Goals and Benefits	When to use
• The objective of an ETL process is to facilitate the data movement and transformation • ETL is the technology that performs three distinct functions of data movement: ○ the Extraction of data from one or more sources ○ the Transformations of the data e.g. cleansing, reformatting, standardization, aggregation, or the application of any number of business rules and ○ the Loading of the resulting data set into specified target systems or file formats • ETL processes are reusable components that can be scheduled to perform data movement jobs on a regular basis • ETL supports massive parallel processing (MPP) for large data volumes	• Data movement across or within systems involving high data volumes and complex business rules • Load data into data warehouses (DWH), operational data stores (ODS), datamarts (DM)
Strategy	**Limitations**
• ETL processes are in usually grouped and executed as batch jobs • ETL tools (like Informatica PowerCenter) are used to implement the ETL processes • ETL processes are designed to be very efficient, scalable, and maintainable	• Real-time event based transfers • Transactional integrations

ETL Architecture Pattern

Most data warehousing projects consolidate data from different source systems. Each separate system may also use a different data organization/format. Common data source formats are relational databases and flat files, but may 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 fetching from outside sources such as through web spidering or screen-scraping. The streaming of the extracted data source and load on-the-fly to the destination database is another way of performing ETL when no intermediate data storage is required. In general, the goal of the extraction phase is to convert the data into a single format which is appropriate for transformation processing.

An intrinsic part of the extraction involves the parsing of extracted data, resulting in a check if the data meets an expected pattern or structure. If not, the data may be rejected entirely or in part.

Transform

The transform stage applies to a series of rules or functions to the extracted data from the source to derive the data for loading into the end target. Some data sources will require very little or even no manipulation of data. In other cases, one or more of the following transformation types may be required to meet the business and technical needs of the target database:

- Selecting only certain columns to load (or selecting null columns not to load). For example, if the source data has three columns (also called attributes), for example roll_no, age, and salary, then the extraction may take only roll_no and salary. Similarly, the extraction mechanism may ignore all those records where salary is not present (salary = null).
- Translating coded values (*e.g.*, if the source system stores 1 for male and 2 for female, but the warehouse stores M for male and F for female)
- Encoding free-form values (*e.g.*, mapping "Male" to "1")

- Deriving a new calculated value (*e.g.*, sale_amount = qty * unit_price)
- Sorting
- Joining data from multiple sources (*e.g.*, lookup, merge) and deduplicating the data
- Aggregation (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.*, putting a comma-separated list specified as a string in one column as individual values in different columns)
- Disaggregation of repeating columns into a separate detail table (*e.g.*, moving a series of addresses in one record into single addresses in a set of records in a linked *address* table)
- Lookup and validate the relevant data from tables or referential files for slowly changing dimensions.
- Applying any form of simple or complex data validation. If validation fails, it may result in a full, partial or no rejection of the data, and thus none, some or all 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, for example, when a code translation parses an unknown code in the extracted data.

Load

The load phase loads the data into the end target, usually the data warehouse (DW). Depending on the requirements of the organization, this process varies widely. Some data warehouses may overwrite existing information with cumulative information, frequently updating extract data is done on daily, weekly or monthly basis. Other DW (or even other parts of the same DW) may add new data in a historicized form, for example, hourly. To understand this, consider a DW that is required to maintain sales records of the last year. Then, the DW will overwrite any data that is older than a year with newer data. However, the entry of data for any one year window will be made in a historicized 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 DW.

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 this data and consolidate it 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 of this would be an Expense and Cost Recovery System (ECRS) such as used by accountancies, consultancies and lawyers. 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

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
10. Clean up

Challenges

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 will need to be managed by transform rules specifications. This will lead 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 analysts should establish the scalability of an ETL system across the lifetime of its usage. This includes understanding the volumes of data that will have to 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 microbatch to integration with message queues or real-time change-data capture for continuous transformation and update

Performance

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. The fastest ETL record is currently held by Syncsort,[1] Vertica and HP at 5.4TB in under an hour which is more than twice as fast as the earlier record held by Microsoft and Unisys.

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 which 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` will significantly (x100) decrease 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. You 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 of their indices can really help.

Another common issue occurs when the data is spread between several databases, and processing is done in those databases sequentially. Sometimes database replication may be involved as a method of copying data between databases - and this 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 allows processing to take maximum advantage of parallel processing. For example, if you need to load data into two databases, you can run the loads in parallel (instead of loading into 1st - and then replicating into the 2nd).

Of course, sometimes processing must take place sequentially. For example, you usually need to get dimensional (reference) data before you can get and validate the rows for main "fact" tables.

Parallel processing

A recent development in ETL software is the implementation of parallel processing. This has enabled a number of methods to improve overall performance of ETL processes 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. For example: 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, for example, 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

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 will 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, and so on.

Virtual ETL

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. So-called 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[2] 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

Keys are some of the most important objects in all relational databases as they tie everything together. A primary key is a column which is the identifier for a given entity, where a foreign key is a column in another table which refers a primary key. These keys can also be made up from several columns, in which case they are composite keys. In many cases the primary key is an auto generated integer which 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 will usually be more than one datasource being loaded into the warehouse the keys are an important concern to be addressed.
Your customers might be represented in several data sources, and in one their SSN (Social Security Number) might be the primary key, their phone number in another and a surrogate in the third. All of the customers information needs to be consolidated into one dimension table.

A recommended way to deal with the concern is to add a warehouse surrogate key, which will be used as foreign key from the fact table.[3]

Usually updates will 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 ware house must keep track of it even though it is never used in queries or reports.

That is done by creating a lookup table which contains the warehouse surrogate key and the originating key.[4] 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,[5] where three selected ones 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 which dimension row to update/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 record time of deactivation.

Tools

Programmers can set up ETL processes using almost any programming language, but building such processes from scratch can become complex. Increasingly, companies are buying ETL tools to help in the creation of ETL processes.[6]

By using an established ETL framework, one may increase one's chances of ending up with better connectivity and scalability. 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 as opposed to writing large programs to parse files and modify data types - which ETL tools facilitate as much as possible.

- SQL Server Integration Services (included in Microsoft SQL Server product line)
- Oracle Data Integrator (earlier owned by Sunopsis)
- Pervasive Software
- Safe Software
- *SAP BusinessObjects Data Integrator* (known as *BusinessObjects Data Integrator* before the acquisition of BusinessObjects by SAP corporation)
- *SAS Data Integration Server* (in the earlier versions known as *SAS ETL Studio* (version 8) or *SAS Data Integration Studio* (version 9)
- SnapLogic;
- Syncsort DMExpress - High Performance ETL

Open Source / Dual-licensed

- Pentaho
- Talend Open Studio
- Scriptella

References

[1] "New ETL World Record: 5.4 TB Loaded in Under 1 Hour - Syncsort" (http://www.syncsort.com/Portals/0/Resources/Solution/DMX_Solution_WorldRecord.pdf)
[2] Virtual ETL (http://itnewscast.com/etl-architecture-and-business-models)
[3] (Kimball, The Data Warehouse Lifecycle Toolkit, p 332)
[4] Golfarelli/Rizzi, Data Warehouse Design, p 291
[5] Golfarelli/Rizzi, Data Warehouse Design, p 291
[6] ETL poll produces unexpected results (http://www.etltool.com/nieuws/2715_ETL_poll_produces_unexpected_results.htm)

Article Sources and Contributors

Extract, transform, load *Source*: http://en.wikipedia.org/w/index.php?oldid=520752426 *Contributors*: 16@r, Ahmedshuhel, Alai, AltiMario, Andy Dingley, Anir1uph, Arcann, Aremgi, Avnjay, Batfly123, Bcrawford, Beardo, Berny68, Blanchardb, BoBaH32, Bonadea, Bovineone, Bruce1ee, Buddyhutchins, CKlunck, Cenarium, Cgfdmc, Chris the speller, Cory Donnelly, CosmoDad, Crasshopper, Cst17, Cyber Dog, DamsonDragon, Dawnseeker2000, Dbush, DePiep, Debgup, Dewwalker, DhirajGupta, Diego Moya, Digisus, Dmccreary, Download, Dpavlis, DragonHawk, Dreadstar, Ebersphi, Edward, Egandrews, Eglobe55, Ehtisham.rasheed, Elf, Emarket, FatalError, Fbdev1988, Founder DIPM Institute, Freek Verkerk, FreplySpang, Frtande, Ggoli, Gharvett, Ghhutch, GimliDotNet, Graeme Bartlett, Grandmasterkush, GregorB, Gscshoyru, Gvimalku, Hoplon, Hu12, Ikan dev, Ikansoftware, Incnis Mrsi, Inter, J mareeswaran, Jamelan, Jan.ahlers, Jay, Jaymishra79, Jesusluisfernando, John Yesberg, Johnbrownsbody, Joinarnold, Jomis, JonHarder, Jtree09, Kadishmal, Karthi acb, Kelbaker, Kennethmac2000, KeyStroke, Kgaughan, Kgobble101, Khalid hassani, Kinghitz, Kjtobo, Kku, Klausness, Kragen, KrakatoaKatie, Kubanczyk, Kuru, Kuteni, Leirith, Levselector, LilHelpa, Limited Atonement, LokiClock, Lotje, Luk, Madman2001, Mandarax, Marek69, Mark Arsten, Mark Renier, Mattpav, MaxSherbinin, Mereman, Michael Hardy, Mikeblas, MonMan, Mr link, MrOllie, Mtfr, Muhandes, NYC sheehy, Naquada, Nbarth, Ndufva, Nielst, Nitpicking polisher, Nsaa, Ocarbone, OlenaSherbinin, Oracleguru, Orange Suede Sofa, Pasquale, PeterCanthropus, Phatwest, Philippe, Playsafe, Pmerson, Pne, Pnm, Psb777, Q Chris, RFMack, RHaworth, Randomran, RedHillian, Reddies007, Rjwilmsi, Rmoore080, Robert Will, Rohit labhe, Ruebencampbell, S.K., Sam Korn, Sarnholm, Sathish jo, Sboden, Scootey, Sd-100, SebastianHelm, Secmail, Seth Ilys, Sgoel.engg, Shaw76, Shehzad.kazmi, Shijaz, Siegler, Sjakkalle, Smittio, Sstrader, Stephenpace, Stevage, Subtleguru, Suhasmallya, Sutanupaul, Syaskin, TFinn734, Talkietoaster-nc, Tanvi.p.goel, Tassedethe, That Guy, From That Show!, The Anome, Thegerf, Theo10011, Thumperward, Todfather, TomRobinson, Triadic2000, Uhai, Ups2000, Urgos, Veyklevar, VincentJS, Vmcburney, Waclawiczek, Warminghands, Wdyoung, Wikiolap, Winterst, Wjhonson, Wwfchina, Xmlguru, Yaronf, Your Lord and Master, Zhenqinli, ZimZalaBim, 605 anonymous edits

Image Sources, Licenses and Contributors

File:ETL Architecture Pattern.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:ETL_Architecture_Pattern.jpg *License*: Public Domain *Contributors*: ETL

License