# ETL in Data Warehouse

ETL is the Extract-Transform-Load cycle in the data warehouse. This is used to combine data from different sources into one consistent form, which is then used in the data warehouse. The data source of a DW is usually an operational (OLTP) system. These systems are SQL based. In a DW we combine multiple of these sources which present with inconsistencies in the data. So to overcome all the issues in data, we use ETL processing.

The data is captured from source system, then loaded into staging area where transformation occurs and then it is loaded into the DW environment, our target system.

---

## Extraction

So the first step is to get data from the source system and transfer to DW

- source system is the operational RDBMS system
- target system is our DW environment

Now there are two types of extraction techniques

- immediate
- deferred

And three types of extraction methods

- using transactional log
- using database triggers and programs
- capturing through source systems

All these 3 methods can be used as immediate or deferred, so we now shall understand the differences. Firstly going with what is immediate and what is deferred

### Immediate

This is that we pick up every change in database at the moment it happens. This may be helpful in some scenarios but usually is quite an expensive operation if there are frequent updates in the OLTP system.

### Deferred

This is that we define a time frame after which the data is loaded into staging area. This time frame may be the end of the day, week or month. This technique is useful when we do not need to preserve the frequent changes but rather require a more summarized result of data. However, keep in mind that though this technique may not be expensive, the changes in data may be lost if not tackled properly. So, to preserve change we may either extract them or preserve history.

**Some points to note when choosing the correct system**

At times we may need to choose the optimal approach for our extraction, which may be dependent on multiple factors. Keep in mind that there can be different update techniques for different tables in the database.

- for some tables we may want deferred update but any changes should also be preserved
    - so we can either use immediate update for change
    - create some form of history that stores all the changes
- Some tables may always require immediate updates, such as real time data that needs to be processed. However, the deferred updates may also be integrated here as at the end of day we may also need to summarize the data.

So, we can have a mix of technologies in addition to standalone updates depending on the scenario we are provided.

---

## Data Extraction Methods

Now, we will discuss the three data extraction methods used to extract data from the source systems and how these can be used in immediate or deferred states.

1. Extract using **Transactional Log**
2. Extraction using **Database Triggers / Programs**
3. Extraction from **Source Systems**

---

### Extraction using Transactional Log

A transaction is a single successful operation performed on the database. Almost all of the DBMS softwares provide the facility of log files.

**Log Files** are the files that do not store data but rather store the events as they happen in real time, so these storage of events is most useful as it stores what changes are made to the database at a particular instant. Now we can use this change to capture the updates to the database. Another advantage of the log-files is that we can roll-back and restore any of the previous states as no data is being actually manipulated. The actual purpose of log files is to recover the data in case of some form of corruption, but we are modifying this to use as an extraction technique.

The point of focus here can be the log files. So there are 2 techniques, and a mixture of both is used:

- increase file size
- create new files

A log file is increased in size until a desired threshold is reached, after which we create a new file. We can also devise custom techniques based on the scenario. For example, in a massively parallel processing environment it may be more feasible to use multiple log files for faster processing rather than using a single log file.

The method of extraction using Log Files is the fastest and best method if available. As it has no performance penalty and none of the additional development cost is required. Most of the DBMS programs come equipped with this feature and most of the DW implementations provide the integration. This even do not have an impact on the operational systems as no live data is being accessed, but rather the separately stored log is being accessed.

Now, let’s focus on how this method can be used in an immediate or deferred state.

### Immediate

This is quite simple as any change to the log file can be recorded by the staging area and the required operations can be performed. If there are no frequent changes to the fields and no history is required, than we can use a simple log file to capture all the required transactional data.

### Deferred

This deferred update however may present some challenges, but we still have two tricks up our sleeve that can be deployed to solve the problem.

So the main issue being how to maintain a history. So, firstly understand that history here is the timestamp. **When an event occurred**. Now, by an event here we may mean an hour or a day. To accomplish this extraction of a particular time frame, we may:

- Have a **Timestamp**

  The easiest way to have deferred updates in a log files is to have timestamps is our file. Most of the log files have the timestamp implementation by default. So, having a timestamp makes things easier. We can extract data based on specific timestamp and this can also be used in comparing and extracting the changes.

- **Compare Files**

  Suppose we have some archival files but have no timestamps so what do we do? Now the simplest method is to use the method of comparing files to extract particular changes. So, to understand suppose:

  - File A has data till today, we extracted this file today
  - File B is the file we extracted yesterday and contains records till yesterday
  - Now, the simplest thing is that we take difference of these two files to get data for today

  But just remember that this may become in feasible when the file size is very large as comparison would be time-consuming, expensive and the worst option in this method.

---

## Using Database Triggers / Programs

A database trigger is a small utility that is linked to a specific table This is like a small function that performs a desired operation when an update happens to a table. This update can be the add, update or delete operation.

We can use the triggers to extract specific information from source system if the log files are not available or if we require any additional information that is not available in the log files.

The only catch here is the additional cost of trigger programs in the source system. These programs need to be written and tested and running a trigger operation on the source system has performance penalties.

Here the trigger design is the differentiating factor between immediate and deferred updates.

---

## Capturing Data from Source System

This option is that we write custom programs to read desired data from source system’s files. This option will be used when the source system is not a RDBMS or data needs to be extracted from a legacy system.

Suppose, that we hav data in multiple json or csv files, now useful information can be extracted by writing a custom program in python that creates some meaningful data. In reality we will write programs but these programs will be running on the source systems as it is a waste to transfer large amounts of raw data back and forth. Rather we can process and extract data to staging area.

This is the worst and most costly method. One is the huge performance penalty and second is the expertise, time and cost required to write and test the programs deployed for extracting this data.

---

### Parallel Extraction

This is the last point that we need to consider when extracting data. Either we need to extract serially or in a parallel fashion. So just understand this that we can have a hybrid model and things are not exactly in a serial fashion.

To fully understand this we need to keep dependence in mind. If a table is dependent on another table than it will be executed in serial manner waiting for the depended table to be extracted before its own extraction can begin. So, this is how serial is executed when one table has to wait for the other to be extracted.

Now, the thing about parallel is that it can be executed if tables are not dependent or even within the table. For example, we can extract two independent table such as students of Lahore and Islamabad campus in parallel. Even considering the table of Lahore only, the table can be extracted in parallel from many nodes to increase performance.

Suppose, courses and student’s registration. The table of courses should be extracted first and till then registration table waits. But once courses are extracted, the registration details can be extracted in parallel. This is how serial and parallel extraction work in harmony.

This not only applies to extraction but underlaying operations as well.

---

---

## Loading

Now that we have captured and successfully transformed the data, its time to load this transformed data into our DW environment. Here again we have some techniques that depend on the situation and scenarios.

Following are the loading strategies:

- Full-data refresh
- Incremental Data refresh
- Trickle-feed / Continuous update

Now before we move to loading, lets summarize some of the operations that happens along loading the data.

- **Data Loading**

  The data is loaded into Dw or a table

- **Index Maintenance**

  This is necessary because at times we have some combined indexes that need to be redone at every refresh to maintain correctness

- **Statistic Collection**

The overall maintained statistics about the data such as count, mean, median and other such things will need to be refreshed

- **Summary Data maintenance**

  Any summarized data such as views will need to be updated according to the new data

- **Data Mart Construction**

  If the raw data is divided into data marts than those will also need to be updated accordingly

- **Backups**

  The proper backup procedures to maintain integrity of data should be followed and re-run if required

---

## Full Data Refresh

This is also called full block slamming and as the name implies, we truncate and rewrite whole table with new data. Essentially, we store old table in staging area and fill our table with updated data.

This strategy is useful when the table size is small or more than 10% data of table needs to be updated.

The complete process is as follows

- remove referential integrity constraints, this prevents errors with other tables and performs a cleansing operation on the data
- remove secondary index specification from the table, here the index references from the index table are removed and new indices are constructed after the data refresh
- We also do use shadow tables while update operation, these allow the queries to be run on previously available data and prevent any downtimes. The extra copies trade storage for availability

## Incremental Data Refresh

Here we append the new data into existing tables. Here we can have two strategies

- Directly load into target table
- first load into a shadow table and than target table

If we load directly into target table than we need to implement some securities in place to prevent bad data

- Indices should be maintained
- redo statistics, summaries and data marts if demographics have changed
- implement a table lock to prevent dirty reads

If we use a shadow table than we are using additional space in the staging area to essentially provide a rollback option incase something goes wrong. Here an advantage is that insert operations from the shadow table will preserve indices and the performance penalty for maintaining indexes is avoided. However, still we are trading storage for performance, so everything is dependent in the requirements.

We will use incremental data refresh strategy when less than 10% data of table is being updated or the table size is very large and whole table can not be put into staging area for full data refresh (now unavailability of storage impacts performance).

Another thing to consider is the update operation, to update is generally 10 times more expensive as any existing data needs to be considered to prevent redundancies in the data. Also in case of secondary indexes, maintaining an index is 3x resource intensive of a single row insertion. So choose wisely.

### Trickle-Feed / Continuous

This approach uses SQL queries to continuously update data. This approach keeps the most up to date copy of data but has huge performance penalties as all of the above batch operations need to be performed on the individual insertions.

However, in scenarios where real time data is required this may be feasible rather than waiting for batch operations. All depending on the requirements.

A hybrid model can also be implemented where system waits for a batch size or time threshold. Suppose the system waits for 10 seconds or a batch size of 5 operations to send data, so whichever expires first triggers the update operation.

---

## ETL vs ELT

| ETL | ELT |
|---|---|
| Extract-Transform-Load | Extract-Load-Transform |
| Performs operations on data and loads into RDBMS databases | RAW data is loaded into RDBMS tables |
| The transformed data is loaded into target or shadow tables | SQL is used to transform the raw data into desired form. (SQL is used because it is designed for batch operations). The data is loaded into DW using the above defined loading strategies |
| Performed using the resources of Source Systems if sufficient resources are available and this transformation results in reduction of data size | Here the Data Warehouse server's resources are used and it acts as a transformation server. |
| If operational system is distributed, high cost per node or sufficient resources are not available then we use a separate **ETL Transformation Engine** which provides the resources needed for transformations. | This is helpful when operational system are lacking resources and we do not want a separate transformation server. However, the DW should hav a large enough staging area to load all data and handle all the SQL queries. |
| | This works well only for more batch oriented applications because SQL transformation is most efficient on batch operations |