# (Big) Data Engineering In Depth

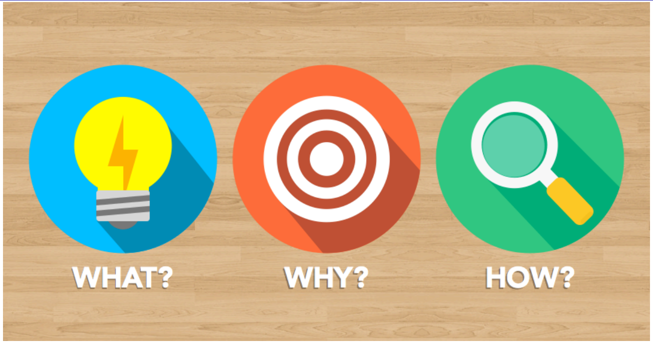
From Beginner to Professional

Moustafa Alaa
Senior Big Data Engineer
MoustafaAlaa Moustafa Alaa @Moustafa_alaa22
Garage Education
mustafa.alaa.mohamed@gmail.com

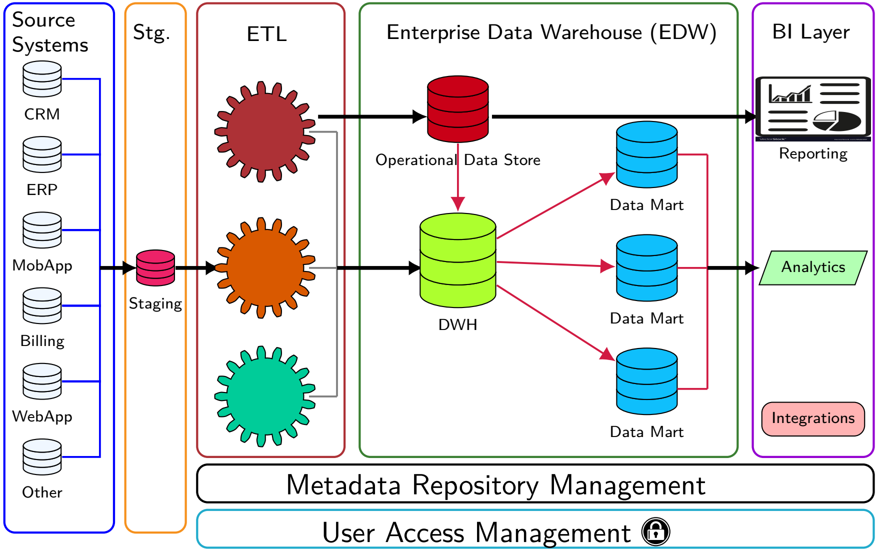
The Definitive Guide to Big Data Engineering Tasks

## Sub-Section: Data Integration and ETL Layer

## Data Integration

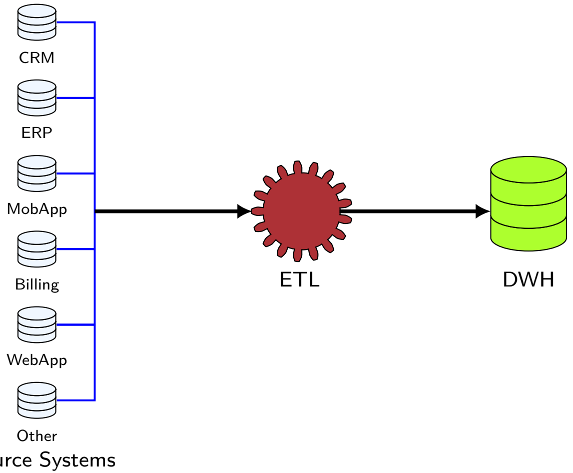


# Data Integration: The Big Picture



## Data Integration: The Big Picture

Data Integration Overview



## What is ETL?

- The ETL is the process of
  - **Extracting**: the data from one or more source system
  - **Transformation**: apply some rules over the extracted data including
    - Cleansing ex: remove null, or trim spaces.
    - Drop duplicates.
    - joining with lookups to validate values or enrich the data.
    - Reshaping or changing the data structure.
    - Adding new columns or removing columns from these data.
    - Change of data granularity.
    - Convert data types.
  - **Loading**: loading the transformed data into the target table based on the required format.

## What is ETL?

- The ETL (Extraction, Transformation, Loading) is the primary core function for any data engineering (DWH) team.
- This team takes the delivered output from the previous stage (data modeling) and start to implement the mapping.
- The implementation of the ETL preferred to be unified across the team members and the organization.

## Why?

### Why ETL become mandatory in any DWH system?

Because of the rapidly increase in the data volumes, and the variety of the data types structure, semi-structure, and non-structure data.

## ETL Characteristics

- Successful ETL design have the following characteristics:
  - ☑ Maintainable.
  - ☑ Reusable.
  - ☑ Well-Performed.
  - ☑ Reliable.
  - ☑ Resilient.
  - ☑ Secure.

## ETL Best Practice

- The keys to most of ETL implementation as following:
  - ✓ **Auditing**
  - ✓ **Managing Bad Data (Rejection Handling)**
  - ✓ **Data Lineage**
  - ✓ **Modularity**
  - ✓ **Logging**
  - ✓ **Error Handling**
  - ✓ **Atomicity**

## ETL: Auditing

## ETL Auditing

- Example
  - We have a billing source system and we need to integrate this source system in our DWH.
  - Assume you get every day around 3M rows of data with total amount around 1M$.
  - Auditing helps us into three parts:
    - Confirm the received rows are inserted into the target or (target + malformed = source).
    - Identify the abnormal behavior if the source row changed maybe 3M (+/- 1%). The same metrics for the target and malformed records to be analyzed.
    - Identify the abnormal behavior in the total aggregations, for example if we have a peak up to the total amount more than the normal (+/- 5%).

## ETL Auditing

- Auditing: The most important key to have a well-designed ETL architecture is to support the auditing row counts, aggregation, and other metrics based on the business and the use case. The first and most important quality gate is based on the auditing.
    - Auditing help identifying the data abnormalities even if the ETL job doesn't have any errors.
    - Auditing implementation differs between systems based on the use case.
    - It is Customized based on the source system's criticality that could add more auditing metrics.
    - We could extend auditing functions to be queryable on database monitoring or dashboards.

## ETL: Logging

## ETL Logging

- Logging: is key to any successful ETL architecture, and it requires implementing a general strategy for all projects.
  - ETL logging includes logs of any activtities or events that occur in the ETL job before, during, and after every stage.
  - Some of the ETL software tools have their logs for the ETL; however, it could extend to other logs added to enhanced the ETL logging.

## ETL Logging

- What types of events or metrics do we need to capture during ETL logging?
    - *Start and stop events.*
    - *Status*
    - *Errors and other exceptions*
    - *Audit information*
    - *Testing and debugging information*

## ETL Logging

- Some consideration during the building of the logging strategy:
  - How to make your logs **format** easy for analysis?
  - Do you have a dashboard or tools for logs visualization, or will it be an ad-hoc fashion?
  - Who is the audience for these logs, and what are their needs?
  - Are there any security requirements or restrictions for data logging?
  - What is the retention policy?

## ETL: Managing Bad Data (Rejection Handling)

## ETL: Managing Bad Data (Rejection Handling)

- In some systems, you get a malformed data, and it needs special handling (not removing).
- It means the data is not accurate or as expected when compared to the same source system.
- If we use this data as-is, we are risk affecting the quality of the business reports.
- To manage the malformed records, we need to apply the following steps
  - Define what is the meaning by well-formed records or accurate record.
  - Configure handling actions for the malformed records.
  - Configure handling actions for the malformed records..

## ETL: Managing Bad Data (Rejection Handling)

- Example
  - We have a CSV file that contains 5 columns, and we need to apply the ETL on this file.

ETL: Managing Bad Data Part 1

```
{
"sourceName": "3G_ERCSN",
"dataFileDelimiter": "|",
"totalInputFileColumns": "5",
"rejectedRecordsPath": "/RAW_ZONE/REJECTION/3G_ERCSN/",
"schemaName": "MOD",
"targetTable": "Singl_KPI",
"saveMode": "Append",
"inputFileFormat": "processing",
"outputFormat": "orc",
"header": "false",
"partitionColumns_3G_ERCSN": "event_date,batch_id",
"getDetailedAuditStats": "false",
"inputSchema": [
{
```

## ETL: Managing Bad Data Part 2

```
{
"columnName": "C0",
"columnType": "IntegerType",
"isNullable": "true"
},
{
"columnName": "C1",
"columnType": "StringType",
"isNullable": "true"
},
.
.
.
{
"columnName": "C4",
"columnType": "TimestampType",
"isNullable": "true"
}
```

## ETL: Modularity

## ETL Modularity

- ETL architecture is not only the quality of data but also how the code is reusable and modularized.
- ETL architecture is not only the quality of data but also how the code is reusable and modularized.
- How to achieve this modularity?
  - Design template jobs for each part of the system.
  - Standard methods and strategy for common tasks such as logging, error handling, and rejecting handling.
  - Reduce the duplicate code by creating common libraries for the ETL.
  - Create common functionality for unit testing.

## ETL Modularity

- How does this affect the team?
    - Maintainability.
    - Debugging.
    - Investigation.
    - Testing.
    - Sharing the tasks between the team.
- This part is a continuous enhancement part.

## ETL: Data Lineage

## ETL: Data Lineage

- Data Lineage is an essential part of our process to understand how this data transformed and where this data originated.
- It helps to make the trust in the data by connecting the target into the source.
- It helps with data tracing at the row level.

## ETL: Data Lineage Example

- We have a CSV file that contains 5 columns and we need to apply ETL process for this file assume the name is 3g_ercsn.csv it could be any name.
- After process this file, we rename this file to be filename_batch_id.

| trns_id | cell_id | cust_number | app | trns_ts | batch_id |
|---|---|---|---|---|---|
| 123 | C15343 | 503987123 | fb | 2020-04-25 12:12:33.4 | 20200426020202 |
| 3436 | C15341 | 503987123 | fb | 2020-04-25 15:12:33.4 | 20200426020202 |
| 43634 | C15353 | 503987135 | twitter | 2020-04-25 18:12:33.4 | 20200426020202 |
| 32632 | C15123 | 503987151 | youtube | 2020-04-25 22:12:33.4 | 20200426020202 |

## ETL: Error Handling

## ETL Error Handling

- Error handling answers a question, What should we do if the process failed?
- It also shows can prevent failure and handle these errors.
- We deal with errors using different strategies:
  - Fail safe (a.k.a. graceful shutdown/stop).
  - Fail fast.
  - Design error handling framework it includes: prevention and response.
  - Design framework dealing with error responses.

## ETL: Atomicity

## ETL Atomicity

- Atomicity refers to breaking more significant parts into smaller individual parts.
- It also refers to how to split bigger ETL process or job into smaller parts.
- Implementation of the atomicity pattern involves the identification of business parts and how to split it into smaller units for proper execution.
- We refer to the anti-pattern for atomicity by monolithic.
- Atomicity requires the ETL engineer to have some overview of the whole project cycle and how we can split it without affecting the business.

## ETL Atomicity

Why do we need Atomicity?

- Maintainability and readability of the code.
- It helps in code tracing, testing, and enhancing.
- Agile development and parallel work.
- Avoid a single failure job, which enhances performance and resource optimization.

## ETL: Paper and Pen Design

## References

- ETL Best Practices By Tim Mitchell