<https://chartio.com/blog/best-practices-for-developing-an-elt-pipeline/>

standard ELT data stack.

* **Extract and load**: [Stitch](https://www.stitchdata.com/) or [Fivetran](https://fivetran.com/) to get data from upstream sources into the warehouse
* **Data warehouse**: [Google BigQuery](https://cloud.google.com/bigquery), [AWS Redshift](https://aws.amazon.com/redshift/), or [Snowflake](http://snowflake.com/)
* **Transformation layer**: [dbt](https://www.getdbt.com/), [Matillion](https://www.matillion.com/), or [Dataform](https://dataform.co/) for managing transforms
* **Business intelligence tool**: [Chartio](https://chartio.com/), Redash, Mode, Sisense, [Looker](https://chartio.com/product/chartio-vs-looker/), [Tableau](https://chartio.com/product/chartio-vs-tableau/), etc. for presenting data to end users
* **Downstream data consumers** (maybe): Customer data platforms, machine-learning models, or other data consumers could fit into this section

chain as a [Directed Acyclic Graph](https://www.astronomer.io/blog/what-exactly-is-a-dag/), or, simply, DAG. It’s “Directed” because it has a direction—from raw data to cleaned data, “Acyclic” because it cannot have transformations that are mutually dependent, and a “Graph” because it can be expressed as a network of nodes and edges

principles I recommend following when developing this system.

**Use schemas to group transformations by type**

3.

# **ETL Best Practices for Optimal Integration**

Optimization:

* Strip out unneccesary data in the begn. Eg redundant entries are cleaned before etl
* Use incremental data updates- only new data added rather than replacing existing data- pretty tricky
* Maximize data quality-data must be clean as possib- automated data quality toolks finds missing, inconsistent data sets
* Automate-clean move and verify results
* Use parallel processing ie, multiple integrations at once. Time-to-value is minimized
* Cache data-keep it in memory/disks
* Establish and track metics
* Workload management to improve etl runtimes

4.

## COPY data from multiple, evenly sized files

Amazon Redshift is an MPP (massively parallel processing) database, where all the compute nodes divide and parallelize the work of ingesting data. Each node is further subdivided into slices, with each slice having one or more dedicated cores, equally dividing the processing capacity. The number of [slices per node](http://docs.aws.amazon.com/redshift/latest/mgmt/working-with-clusters.html) depends on the node type of the cluster.

When splitting your data files, ensure that they are of approximately equal size – between 1 MB and 1 GB after compression. The number of files should be a multiple of the number of slices in your cluster.

## Use workload management to improve ETL runtimes

* Create a queue dedicated to your ETL processes. Configure this queue with a small number of slots (5 or fewer). Amazon Redshift is designed for analytics queries, rather than transaction processing. The cost of COMMIT is relatively high, and excessive use of COMMIT can result in queries waiting for access to the commit queue. Because ETL is a commit-intensive process, having a separate queue with a small number of slots helps mitigate this issue.
* Claim extra memory available in a queue. When executing an ETL query, you can take advantage of the [wlm\_query\_slot\_count](http://docs.aws.amazon.com/redshift/latest/dg/r_wlm_query_slot_count.html" \t "_blank) to claim the extra memory available in a particular queue. For example, a typical ETL process might involve COPYing raw data into a staging table so that downstream ETL jobs can run transformations that calculate daily, weekly, and monthly aggregates. To speed up the COPY process (so that the downstream tasks can start in parallel sooner), the wlm\_query\_slot\_count can be increased for this step.
* Create a separate queue for reporting queries. Configure query monitoring rules on this queue to further manage long-running and expensive queries.
* Take advantage of the [dynamic memory parameters](http://docs.aws.amazon.com/redshift/latest/dg/cm-c-wlm-dynamic-memory-allocation.html). They swap the memory from your ETL to your reporting queue after the ETL job has completed.

## Perform table maintenance regularly

columnar database, which enables fast transformations for aggregating data. database tables regularly are VACUUMed and ANALYZEd. Amazon Redshift uses a cost-based query planner and optimizer using statistics about tables to make good decisions about the query plan for the SQL statements.

## Perform multiple steps in a single transaction

ETL transformation logic often spans multiple steps. Because commits in Amazon Redshift are expensive, if each ETL step performs a commit, multiple concurrent ETL processes can take a long time to execute. steps in an ETL script should be surrounded by a BEGIN…END

## Loading data in bulk

 store and query petabyte-scale datasets. Use [temporary staging](http://docs.aws.amazon.com/redshift/latest/dg/merge-create-staging-table.html) tables to hold the data for transformation. These tables are automatically dropped after the ETL session is complete.

## Use UNLOAD to extract large result sets

Fetching a large number of rows using SELECT is expensive and takes a long time. When a large amount of data is fetched from the Amazon Redshift cluster, the leader node has to hold the data temporarily until the fetches are complete. Further, data is streamed out sequentially, which results in longer elapsed time. As a result, the leader node can become hot, which not only affects the SELECT that is being executed, but also throttles resources for creating execution plans and managing the overall cluster resources.

5.

6. <https://www.fivetran.com/blog/building-efficient-data-pipelines-with-incremental-updates>

**Building Efficient Data Pipelines With Incremental Updates**

API data sources may not have changelogs, but may instead have timestamps that indicate when a record was last changed. The goal is to identify, extract, and load records that have been added or updated since the last sync.

These updates may be whole-row, in which a record’s unique [primary key](https://fivetran.com/blog/building-an-idempotent-data-pipeline) and all values are tracked, offering a full snapshot of each record. They may also be partial-row, in which only a primary key and changed values are tracked.

7. Azure Databricks provides different cluster options based on business needs:

Autoscaling – Databricks has an auto-scaling feature, which can help with scaling. As the workload increases more nodes will be spun up to accommodate the workload. This is especially good for SQL Warehouse, and ETL workloads. It probably shouldn’t be used for machine learning workloads that are very iterative. The cluster page has a min and max worker node settings. So, you might start your testing with a min = 4, max = 15. As the workload diminishes the workers are deallocated and you are not charged. You can review the executors tab or cluster page in the spark UI to see how many workers are spun up in your workload/test.

Data partition – Data and the partitions of the data can greatly affect memory consumption and performance. If one partition is skewed it can cause OOM on a worker on shuffle operations.

Partition the data according to the date or key columns and run these partitions sequentially with lesser cluster configuration. For example – If my data volume is 2,000 GB, I can choose a cluster with General Purpose–Standard\_D32S\_v3 128 GB RAM 32 cores 6 DBU 10-20 nodes or I can partition my data to eight partitions having 250 GB each and have cluster size as Standard\_D32S\_v3 128 GB RAM 32 cores 6 DBU 1–3 nodes. This will reduce the cluster cost to a greater extent.

Maximum RAM size that can be used in Databricks cluster is 432 GB and maximum number of nodes that can be allocated is 1200. The number of nodes to be used varies according to the cluster location and subscription limits.

Larger memory with fewer workers – In Spark Shuffle, operations are costlier and it will be better to choose larger memory with fewer workers. Larger memory and smaller number of workers will make the shuffle operations more efficient and reduce OOM issues.

Other activities in worker nodes – When you are choosing the worker nodes have some additional memory for the operating system, Spark dll/jars and various other activities.

To reduce costs, partition the data and use cluster configuration with a smaller number of nodes.

9.

It has been hypothesized that by extending parallelism on both source and target

side and by replacing lookup operations which use extensive memory to cache

data by joiner operations on sorted data will optimize the ETL process

rearrangement of logical transformation in order to

improve performance and maintain data accuracy

they have further extended the notion of state space

search optimization technique by devising a method based on the specifics of an

ETL workflow that can reduce its execution cost either by decreasing the total

number of processes or by changing the execution order of the processes. The

specification and design of ETL workflow are the prime focus to be kept to

optimize the performance

Then, we introduce greedy and

heuristic search algorithms to reduce the search space.

He proposed

ETL management through metadata management. A metadata management

system with good design can highly improve the ETL efficiency. However, the

large amount and the wide distribution of metadata in ETL processes cause the

mismanagement of metadata. To deal with this problem, they suggest intensively

manage ETL by metadata repository. Metadata repository can show DBA the

metadata in a direct, simple, and centered way, and makes metadata more easily

to understand, therefore metadata management becomes more direct, simple, and

centered.

Another method focussed on static partitioning and parallelization transform level

using multithreading and shared cache in ETL data flows proposed by