**Batch Jobs in Financial Domain**

**Introduction**

“Batch Job” terminology came in from the days of our mainframe computer era wherein the user used to enter programs using punching cards. The end user would provide a batch of programmed cards to the respective systems operator who in turn would be responsible in feeding the information to the computer system.

There has been lot of modifications and alterations on the usage of batching these latest years and new pathways have opened to perform these tasks. Though its core background process will never cease to exist. Starting from routine organizational process, like payroll or to maintain the servers from getting tied up throughout the workday, these batch jobs play an important role for many years to come.

Keeping this in mind, this entire document has been prepared to introduce the end user to the process of managing batch jobs, keeping in mind on how they will be implementing it for their respective organization and employee workflows.

**How does Batch Processing Works?**

In layman terms, a batch job is a scheduled automated program assigned to be executed on resource/computer without any manual intervention. These are often queued up during working hours and triggered during the evening or weekends when the systems are in idle state.

Once the Batch job is submitted, the respective job with its unique job ID enters a job queue where it needs to wait until the system is ready to process the job. If the job queue contains multiple jobs in pending to be processed state, the batch system processes the jobs either in chronological order or by priority. The above said jobs are structured in tree format. These are normally used to automate tasks that showcases regularity like payroll, but not necessarily it needs to be executed during the day or have any employee/end user interact with the system.

These jobs that are required to be executed on daily/weekly/monthly basis are generally incorporated into batch schedulers. In some server systems the batch jobs execute in the background while other interactive programs get executed in the foreground, providing these interactive programs priority over batch programs. This is being performed so as to allocate more memory to them instead of foreground applications. The high volumes of data that these batch jobs are processing correctly justifies the consumption of long-term memory if executed in foreground. The most important and unmissed benefits or batch jobs include:

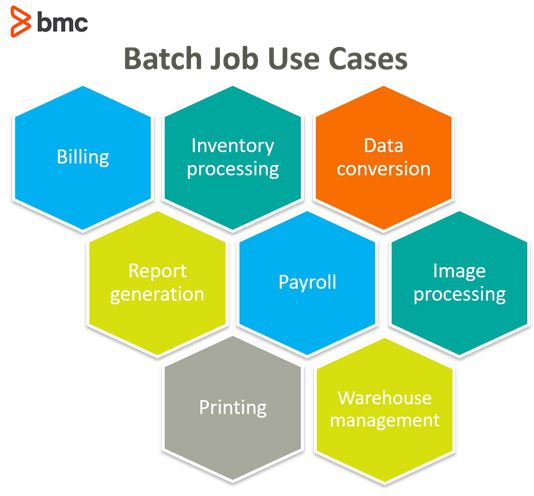
* Huge sized programs and applications can utilize more dedicated servers when the work processes are in non-daytime mode.
* With very less users working during off hours, the overall performance will definitely be faster and more efficient during this time. During working hours, it could be restricted to less servers and count needed.
* Users are freed up to focus on less repeatable, and more innovative tasks.

**Batch Job Scenarios and Relevance**

The Legacy batch job setups are still highly relevant and still in use in almost every business computing environment till this day despite the advancement and modernization in our technologies.

A telecom billing application is a perfect example of a batch job. Initially the application records the phone call details from the enterprise information system. Then, the application generates a monthly bill for each customer account. This type of activity can execute without anyone’s intervention.

Other common usage of batch jobs include:



**Steps in a batch job**

A step is a self-reliant and serial phase of a batch job. Batch Jobs are an inclusive package of both chunk-oriented steps and task-oriented steps.

**Chunk-oriented steps:**

1. The data is read and processed one item at a time from a specific data source, such as files in a directory or entries on a database.
2. Business processing manipulates each item using the business logic defined by the application, like formatting or filtering.
3. The results are grouped into a chunk, which are stored once the chunk reaches a configurable size.

This type of processing typically makes storing results more efficient and helps facilitate the limits of the transaction.

**Task-oriented steps:**

Chunk steps often process large amounts of data and in turn, can take a long time to complete. Utilizing batch frameworks, these chunk steps can then bookmark their progress using checkpoints. Using the input retrieval and output writing parts of a chunk step, the current position is saved after it is processed. Should a certain chunk step be interrupted, it can be restarted from the last checkpoint.

**Parallel processing**

For batch jobs that require large amounts of data to be processed or for computations involving a very large number of independent computations, parallel processing may be beneficial, reducing the total number of jobs submitted.

Parallel processing is typically most beneficial in two scenarios:

* Steps that don’t depend on each other and can run on different threads.
* Chunk-oriented steps where the processing of each item doesn’t depend on the results of the previous one.

Batch frameworks provide mechanisms for developers to define groups of specific independent steps, splitting chunk steps into parts that can then run in parallel.

**Decision elements:**

In addition to steps, batch jobs can also contain decision elements. Decision elements utilize the exit status of the previous step to determine the next step or to terminate the batch job altogether. Decision elements set the status of the batch job when terminating it, noting if it was terminated successfully, it was interrupted, or it failed.