brillio

Managing Data Ingestion with AWS Glue Incremental Keys

As data flows from one source to another, incremental data ingestion is one of the most common requirements.

We often hear from our clients:

"Would there be a date-diff mechanism for capturing the newly inserted records?"

"Do we need to overwrite data every time? If yes, then as data grows, how would we manage the huge data volume?"

"How can we just ingest the fresh / new records daily?"

"Our ETL tech-stack (Talend / Informatica / Glue / Stremasets) is the most expensive service. Can you help us minimize data processing costs?"

The answer to all the conundrums lies in designing a mechanism to highlight the newly inserted/updated records. Hence, this necessitates the identification of certain attributes in our data that would uniquely highlight the same i.e., the creation of **Incremental Keys.**

What is an Incremental Key & Why Do I Need One?

An incremental key is a field or a combination of fields that specifies the update time of a file or a set of rows that signals the recently modified rows in comparison to others. This field can be used to collect the updated new data rows, saving vast amounts of time in the collection process and improving the performance.

Incremental keys are great for swift update times. The following are the benefits of implementing incremental keys with your data (Database, Data Lake, or Datawarehouse):

- **Shorter update times-** Only updates the newly modified/inserted records.
- **Optimized data storage cost-** Helps reduce redundant data records.
- **Seamless data management-** Easily store the historical records, ensuring the traceability of data operations.

In a pragmatic implementation having a single field often produces redundant records. Hence, it is advisable to select the incremental key as a combination of fields. There are multiple use cases for the incremental key implementation across industries:

Use Case I- Data Migration to Cloud

If the client's ecosystem is a hybrid architecture, some data resides on-premises and the other part on the cloud. In this case, incremental keys are often used to ingest the newly inserted/updated records.

Imagine daily inserting the records for certain entities on the cloud, every time with a complete bulk upload for terabytes of data. This would not only be a tedious but also an expensive process. Therefore, doing a bulk upload for the first time and then just inserting the newly modified records seems more rational. The incremental keys are used for signaling the changed records.

Use Case II- Seamless Living in Cloud

Specific to the downstream analysis of data (analytical requirement or transactional requirement), it is not feasible to load the entire data in the downstream services. Hence, clients are keen on designing a mechanism to capture the daily updates rather than an overwrite mechanism.

Sometimes the primary key or even the combination of keys in a dataset fails to identify the records uniquely. In such cases by inserting a new attribute such as 'update', the timestamp could be used to identify the newly modified records.

Use Case III- Performance Optimization

We have seen with multiple clients that a data processing job would take around 5-6 hours to complete, despite the high compute provided to the ETL service.

In such cases, the only solution left is to improve the data design mechanism by processing the records which were not processed earlier or skipping the redundant data records.

Specific to data commute within AWS environment for data transfer from Redshift / RDS / any other data store to S3 or any other downstream consumption service, the following are the common approaches:

- Unload query with a relevant filter condition
- A PostgreSQL script (or the native data store SQL version) with a relevant filter condition
- Glue ETL job, etcetera

For the initial two approaches (i.e., unload query & a customized script) a filter condition is manually placed to capture the incremental records, but there is no implicit/in-built incremental key.

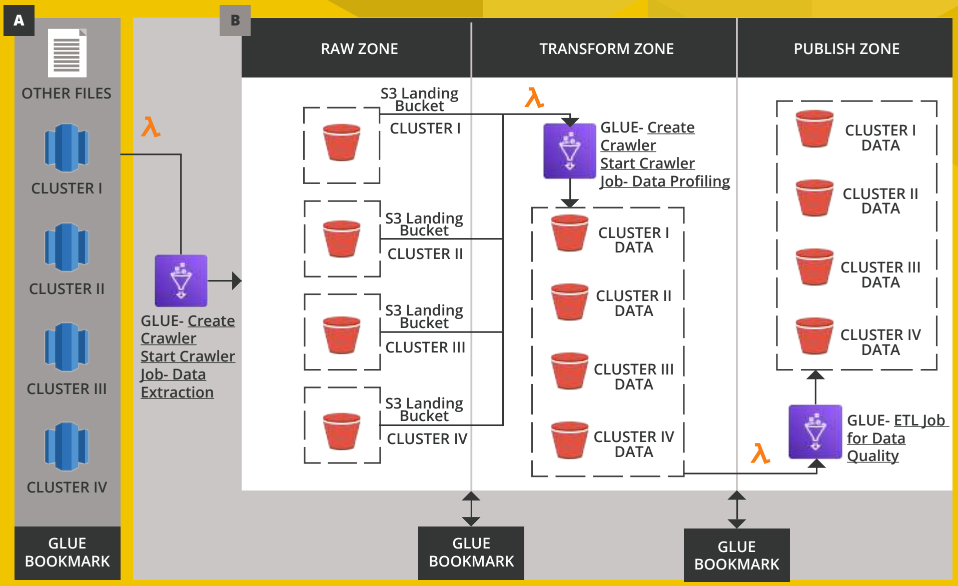
Whereas, with AWS Glue you can design an automated incremental key logic i.e., AWS Glue Incremental Key. AWS Glue's Spark runtime has a mechanism to store the state. This mechanism is used to track data processed by a particular run of an ETL job. The persisted state information is called job bookmark. The following sections would deep-dive on the same.

How to Implement an Incremental Key Using AWS Glue?

With AWS Glue we can design logic to pass multiple fields/attributes as the incremental key. There is a feature in AWS Glue, referred to as job bookmarks. The same needs to be enabled to capture the information for the last processed records with respect to the specified incremental key. We will understand its implementation by taking a sample use case.

Use Case: Within an AWS environment, implementing ETL design from a JDBC store (Redshift / RDS) as the source & S3 as the target, for incremental data ingestion.

The following is the indicative architecture for the sample use case:



Step A: Incremental load from JDBC store (Redshift) to S3

- Metadata (schema) creation for the input data sources in the Glue data catalog via Lambda.
- Glue ETL script- This will consist of the logic for enabling the Glue job bookmark on specific datasets & specifying the incremental key field or the combination of fields.
- As per the business schedule requirement, a lambda would trigger the Glue ETL job for data transfer.

Step B: Incremental data processing within S3 zones

- Metadata (schema) creation for the input data sources in the Glue data catalog via Lambda.
- Glue ETL script- This will consist of the logic for enabling the Glue job bookmark on specific S3 files and the incremental key would be the timestamp of file creation in S3.
- As per the business schedule requirement, S3 event notification would trigger the Glue ETL job for data transfer.

Outcome: This approach will keep a track of already processed data during a previous run of an ETL job by persisting state information from the job run.

What Are the Limits of the AWS Glue Incremental Key (the Break-even Point)?

A POC (proof of concept) is done for Glue incremental ingestion from a JDBC store (Redshift in our case) as the data source. As a part of the POC, various test case scenarios have been designed to identify the limits & scope of using Glue bookmark keys for incremental data ingestion. The following is the outcome of the various test cases:

No.	Test Case	Result	
		Primary Key (Single Field)	**Compound Key (Combination of 2 Fields)**
1	The fields in the incremental key are monotonically increasing.	Pass	Pass
2	The fields in the incremental key are monotonically decreasing.	Pass	Pass
3	The fields in the incremental key are monotonically increasing or decreasing, but have some gaps between them.	Pass	Pass
4	An update using the incremental key on the records.	Fail	Pass (As for the second field value was monotonically increasing)
5	Insertion of values for the gaps or breaking the monotonically increasing/decreasing oder.	Fail	Fail

Thus, a rational selection of fields (compound key) as the incremental key would be preferred over a single field / primary key. Even after using a compound key, if the data in scope does not have a single field that guarantees order, other mechanisms like unloading queries (for redshift) with a DateTime filter condition must opt for data ingestion.

About Author

Balvinder is an Archvisor (Budding Architect & Advisor) in the Cloud Transformation & Data Engineering domain. Driven with tech-no-functional skillset, he is client-centric in his approach and holds experience of working on three cloud hyperscalers (AWS, Azure & GCP). He holds around 3.5 + years of experience in data on cloud consultancy across Financial, Telecom and Healthcare industries.

ABOUT BRILLIO

At Brillio, our customers are at the heart of everything we do. We were founded on the philosophy that to be great at something, you need to be unreasonably focused. That's why we are relentless about delivering the technology-enabled solutions our customers need to thrive in today's digital economy. Simply put, we help our customers accelerate what matters to their business by leveraging our expertise in agile engineering to bring human-centric products to market at warp speed. Born in the digital age, we embrace the four superpowers of technology, enabling our customers to not only improve their current performance but to rethink their business in entirely new ways. Headquartered in Silicon Valley, Brillio has exceptional employees worldwide and is trusted by hundreds of Fortune 2000 organizations across the globe.

https://www.brillio.com/

CONTACT US: info@brillio.com

infor@brillio.com