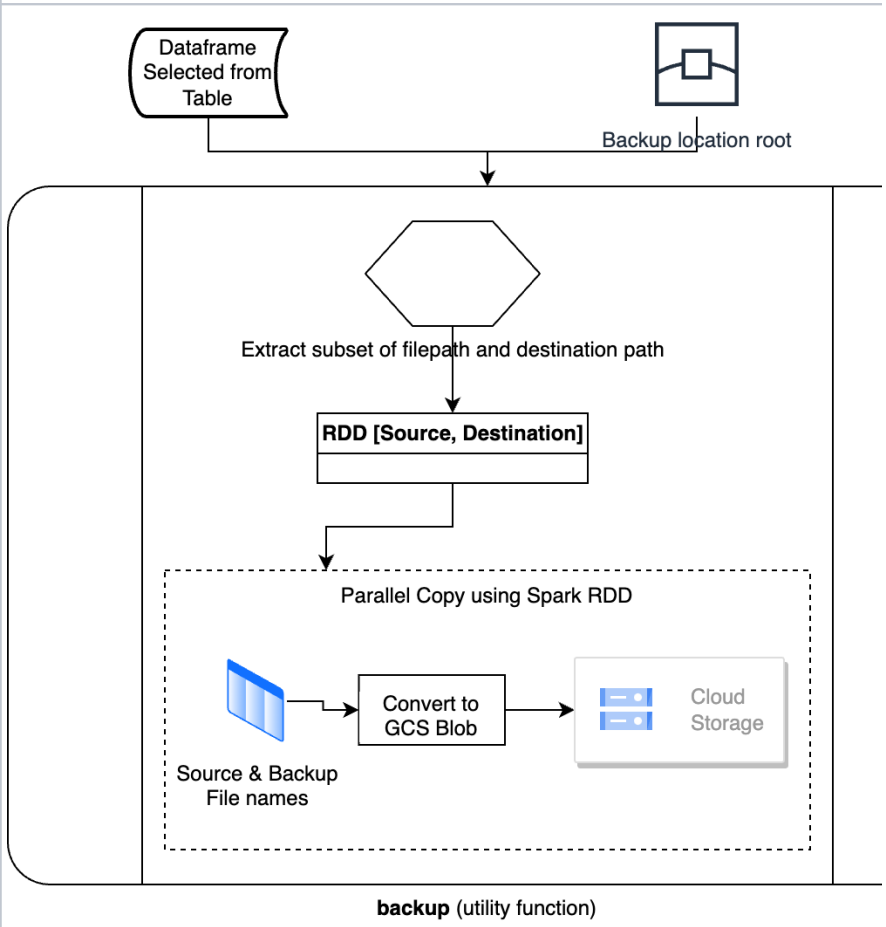
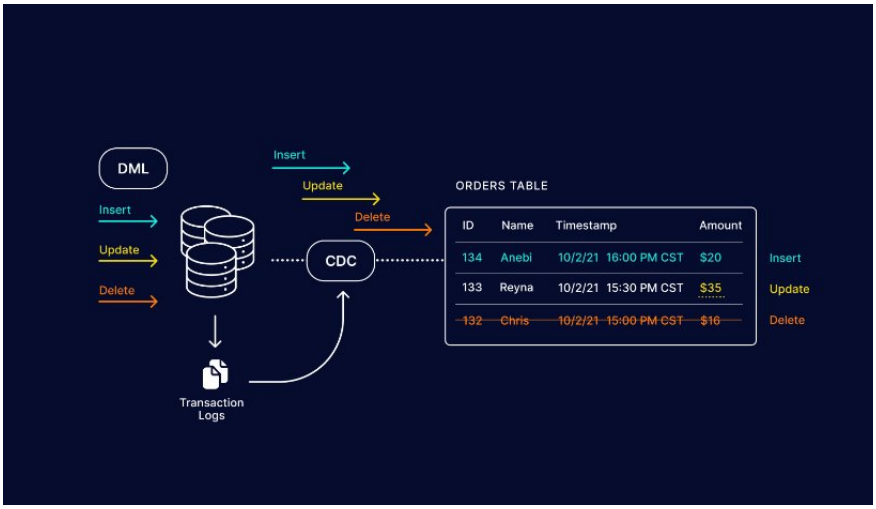
DV Utilities

Repository

https://gecgithub01.walmart.com/dsi-dataventures-luminate/cperf-datapipeline-utilities.git

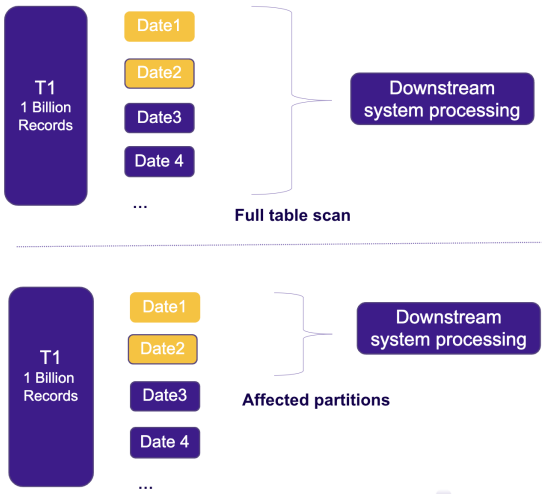
Utility Function	Problem Statement	How does this utility helps in Channel Performance Luminate Data foundation?	How can this be adapted for different use cases across Data ventures & Walmart in other Data pipelines?	Flow / Design Diagram
ETL package				
backup	Problem: For every data pipeline, Table data backup is critical & task repeatedly performing task. For example - In case of defensive measure before overwriting published data and before performing change data capture. Efficient backup sub-routine is highly demanding utility.	Use case overview: There are 2 use cases majorly - 1. We introduce an utility for creating exact copy of the Table data for security, recovery & maintaining standards of compliance 2. To use backup data as source to perform logical computation & not on actual production (published) data Example: 1. If any data discripency is reported, until team identifies root cause we can quickly publish backup data unblock availability of data 2. Using copy of production data (backup) **to perform logical computations viz. Change data capture**	Overview: 1. **backup** utility is an independent function, doesn't require extra setup 2. It qualifies all due dilligence points to be considered before performing backup 3. The **USP of this utility function** is, it doesn't use traditional Spark SQL operation to perform backup, instead it goes at the root of the file system (any cloud filesystem Azure /AWS /GCP) & performs backup at file level by leveraging spark's parallelism	

performDelta	Problem: We are aware data is changing continuously & rapidly in both master data (dimensions) & transactions (facts). Example: 1. Price margin or Cost of an item is updated. 2. An item is newly introduced into the system (new Walmart Item Number) 3. An item is become obsolete & it is removed from system Data warehousing's critical problem is capturing row (record) level changes since the last table refresh execution. It becomes challenging if there is no timestamp audit field in the table from where data is sourced, and if the team that owns that table doesn't communicate about changes. How can we identify the changes without dependening on source team?	Use case overview: 1. Sales, Inventory transactions & Store, Parent Company & Vendor related master data are changing over time, including back dates (history) 2. We introduce an utility which follows best practice of Data warehousing to identify Change data capture 3. On changes, And label every record about Inserting, Deleting, Updating or No change at row (record) level. 4. This helps downstream to depend on a trusted audit column which has label about Insert, Update, Delete & No change then perform their action in downstream system	Overview: 1. **perform Delta** utility is an independent function, doesn't require extra setup 2. It qualifies best practices of Data warehousing concept Slow Changing Dimension (SCD) type 1, meaning - Insert /Update /Delete a record as it is done in source. 3. It required 2 datasets, one previously processed data and a new dataset which is expected to be changed at row level 4. This reusable function adds a new column called **delta_flag** with labels I, U, D, NC 5. **USP of this utility function** is it is so adaptable that it allows you to skip field that are to be ignored during identifying row level changes.	

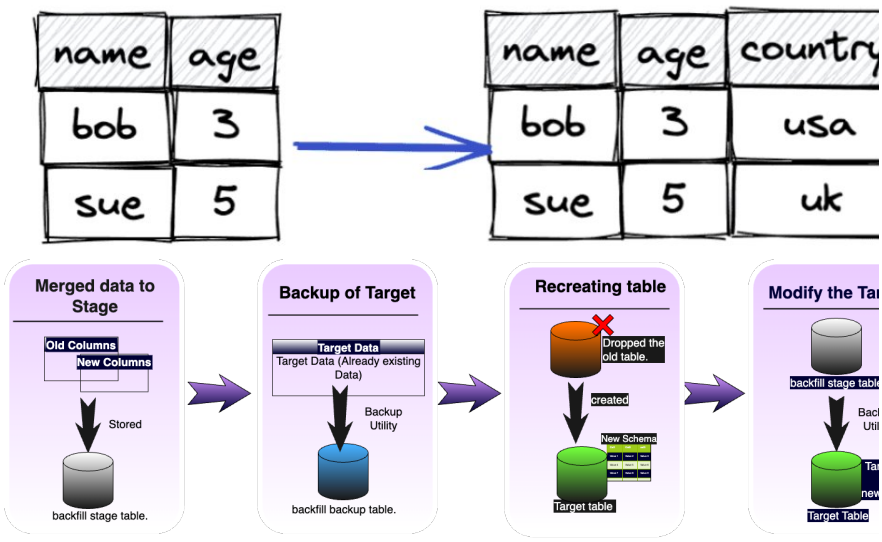
performDeltaUsingChecksum	Problem - With the adoption of delta formats, 2 layer approach and the new column level changes(i.e md5_hash_id , dv_del_ind) , a new method was to be created which would keep track of the changes of the incoming data w.r.t to target . We had to build a solution which could keep track of which records got added , removed or changed , which could be achieved by updating the hash column and delete indicator column based on the changes.	Since now the delta flags are now populated in the consumption layer , it was important to populate such metrics which in our application layer such that in consumption we could easily flag out the records and populate the delta flag as I,U, D or NC accordingly . Now here the columns md5_hash_id and dv_del_ind helped which we populated in our pipeline taking help with this method.	Overview: 1. **performDeltaUsingChecksum** is an independent function and doesn't require extra setup. 2. It keeps track of records which got changed , updated and deleted by updating the the column **dv_del_ind** and **md5_hash_id** columns. 3. It requires 2 datasets , one previously processed data and a new dataset which is expected to be changed at row level and additionally it also takes Primary keys as a parameter which is used as join condition .	
Data Quality package				

executeDQ	Problem: Data accuracy & integrity checking in Data pipelines is important.	Use case overview: 1. After Ingestion of data from source tables, applying quality rules viz. accuracy & integrity helps in quick & correct transformations 2. After performing business transformations & any data warehousing aspect on one or many source tables, applying rules to identify integrity & accuracy of data Examples - 1. Natural / Primary Keys checks 2. Uniqueness 3. Not NULL 4. Business rule as Custom query 5. Schema validation	Overview: 1. GDAP / GDP platform has given us platform to execute quality rules, publish the metrics on GDP Portal & create DQ rules on Portal. 2. **USP** of this utility function **execute DQ** is it avoids all pre-defined steps in pom.xml plus instantiating rule engines, registering & fetching rulesset id & executing rules on dataset 3. Another cool feature is it's implicit nature to be enabled on Dataframe or Dataset [Row] datatype in Scala-Spark	
generateDQ Report	Problem: Data quality rules checks need to be socialized proactively for better & quick support / resolution during failure situations.	Use case overview: 1. For quick & proactive support, resolutions socializing quality rule runs in crisp, meaningful manner report is used in all the Channel performance Data pipelines 2. Email also can contain additional information along with DQ rule runs metrics published, about that pipeline execution	Overview: Generate data quality reports just by specifying ruleset run metrics. **USP of this utility** is HTML / MIME types are supported, adds extra sophistication to reports	

auditDQReport	Problem: Data quality rules checks need to be audited for future refrences and they need to be organized so as to fetch them quickly when required.	This is in design phase, we shall update soon	This is in design phase, We will update soon	
Table package				
getTableLocation	Problem: Most of the spark / hive tables are external, a metadata is mapped to an Cloud Storage location. The location is important because it has physical data files. This location / path trimmed & case sensitive string is important in many places throughout session.	Use case: All tables are external: For safety data & schema (metadata) of table are isolated. In situations like moving physical files of table data requires location / path. Mainly for backup purpose.	Overview: Generic function works in any spark-scala datapipeline to fetch path of table's physical files	

getAffectedPartitions / getAffectedPartitionDates	Problem: Change data capture has become easy with above **perfomDelta** reusable utility. But with data volume comes more challenge. Because **performDelta** is a SQL operation at row/record level. More volume is vulnerable to unexpected failures & poor performance. What is the way to reduce the volume of data in a common situation where most of the data is unchanged only data belonging to few partitions have changes. Example: 1. Sales data is partitioned on business date and transactions are at date level 2. Changes viz. New records, Updated records & Deletes are date level and date is a partition Avoid the partitions which never changed which will cut out unnecessary scan of those date partitions	Use case: Sales (Ecommerce & Stores) source data from various teams FD, Crew, Replinishment refresh data during their restatements. Some team maintain affected partition dates as a table and few teams don't. Identifying which business dates were restated is challenging so, as a source of truth we in Channel Performance identify from Table's physical file's updated timestamp property & capture the business date to filter data.	Overview: Reusable Independent function identifies which date partitions were impacted after the data refresh in the table. This function leverages spark's parallelism and Google Cloud Storage SDK to run through Table's location in hierarchical manner & fetches updated timestamp of a file. **USP of this utility** is it uses ~~spark with~~ GCS SDK in fusion Drawbacks & potential improvements: ~~Dependent on Spark (this can be avoided & use just GCS SDK)~~ ~~Scans full table before identifying impacted partitions after previous data refresh in a table~~ To do: - [x] Remove dependency from spark	

Partition. drop	Problem: Tables are partitioned for better filter ability which has significance to improve performance as it cuts out partitions that are not required during scanning the table. Dropping partition is necessary because tables are external as a reason if we delete physical data file, metadata of table needs to be updated with drop partition	Use case: Partitions are hierarchical. Because tables are external, if physical data is deleted/added the metadata must be in sync with physical data. Deleting huge table is easy by deleting underlying files, to make metadata in sync drop partitions is used.	Overview: Independent function to drop partitions of table. Note: If table is external drop partition doesn't delete the files	
Partition. delete	Problem: Tables are partitioned for better filter ability which has significance to improve performance as it cuts out partitions that are not required during scanning the table. Dropping partition is necessary because tables are external as a reason physical data file is not deleted if only partition is dropped. Because partition drop will update only metadata of the table	Use case: Partitions are hierarchical. Because tables are external, if metadata is deleted/dropped the physical files remain & they must be in sync with metadata. Deleting huge table is easy by deleting underlying files, this function deletes physical files of table	Independent function to delete physical files in partitions of table. Note: If table is external drop partition doesn't delete the files so delete function is necessary	
getSchema	Probelm: Can DDL be parameterized with schema, table-name & it's physical location? Then interpolate these parameterized values in DDL & get the correct DDL string?	Use case: Injects Table, Schema & Location parameters of table into DDL string & returns. Used to create tables if not exists	Overview: Independent function supports all Hive DDLs execution given Table's name, schema & location / path	
Table Schema package: Schema Evolution Automation Reusable functions				

isSchemaChanged, newColumnsAdded, newColumnsDf	Problem: Identify new fields added into transformation and create a dataset with natural /primary keys & all the new columns added. Execute transformation with new columns added / existing columns subtracted goes through pre defined steps.	Use case: Dimensions & Facts of channel performance are evolving according to End user (supplier) requirements. The scope of adding / subtracting column will be expected until the Data Model & Channel Performance System becomes stable. Schema Evolution reusable function help in smooth transition of newly added columns & deleted columns with history data of newly added columns	Overview: For all those data pipelines / tables that have frequent changes in schema anticipated / unexpected and require smooth automation steps to refresh the data. **USP of this utility** is merging new & old table's steps /algorithm are pre-defined & automated.	
getSchemaMap	Probelm: SQL Dialect is different compared to Programming Syntaxes. Can we quickly hop between 2 dialects easily at attribute level to access the properties of column/field?	Use case: After all business transformations & aggregations the columns are not in expected order according to table, getSchemaMap is used to organize columns with the table schema	Overview: Columns & Data types of a table are stored as Map data structure hence helps in accessing the column & it's datatype with O(1) time complexity	
Schema Evolution	Pain points: • Manual intervention during schema update • Multi-Step implementation • Manual Data Backup • Difficulty in History backfilling • 2 complete days to implement.		• Scaling the schema to any extent • Automated Data Backup into the stage tables • Newly incoming data checks. • History backfilling with ease • Independent runs without impacting daily jobs • Human errors are avoided • End-to-end automation of all required tasks. • Productivity improvement.	
Date Time Operations package				

yearMonthDate, yearMonthDateTs, tz Interpolators	Problem: As a Data Engineer it's always helpful if a short / handy interpolators are available for creating dates & timezones	Use case overview: 1. Support we need a run date as java. timestamp. LocalDate type, we don't have to go traditional way of instantiation of class-object. 2. An interpolator is used to create Date, Timestamp & Timezone with static types. 3. Used across channel performance data pipelines for Business Date, Run Date, Start Date, End Date (s)	Overview: **USP of these utilities - These are short hand interpolators, reduces 95% of code**	
getDatesBetweenAs [Generic Type]	Problem: Given range of dates minimum & maximum date, Generate list of all the dates falling in that range inclusing of minimum & maximum. And the datatype of dates can be generic at compile time? The datatype must be specified by user.	Use case overview: 1. To refresh history data for a given date range 2. Given range of date this utility computes all dates that are in the range 3. Those list of dates are added as filter on source data	Overview: Generate all the dates falling in given range inclusive of minimum & maximum. **USP of this utility** is the return type of date is generic, and can be specified at compile time by Data Engineer	
Google Cloud Storage & Google Cloud Services package				
blob, blobs, buckets, bucket, prefix Interpolators	Problem: Google cloud storage is a special file system. It is shared access storage system having hierarchical structure files are organized logically in directory trees for intuitive access. Can the access to the files & file properties be adaptable to programming abstract class & as variables?	Use case: Traversing through hierachical GCS file system in Channel Peformance Data pipelines is applicable in case of identifying Affected partitions of source table, Deleting files in case of table delete etc.	Overview: An interpolated extension to GCS SDK's Blob, Bucket & Page objects. A GCS String can be used as static, strict & strong typed variable **USP of this utility is reduces 95% of code**	

GCSObject	Problem: GCS is a special file system having hierarchical structure. It also has a protocol for accessing files / folders using URI **gs://** The URI has 3 parts - 1. Protocol gs:// 2. Bucket 3. Folder prefix or File path Can we have an singleton object of the above 3 parts to be accessible & injectable in various other data structures programmatically?	Use case: The main purpose of GCSObject is GCS Filesystem has it's unique concepts of file path viz. bucket, blob. Organizing all attributes having 1 single ton object is purpose of this utility GCSObject can be used for GCS actions	Overview: Can be used as singleton object of case class. Organized parts of a GCS URI **USP of this utility is**, conversions are allowed between GCS SDK static types and scala native types	
parallelCopy, parallelRemove, parallelMove	Problem: GCS SDK has Copy, Move, Remove are functions. But can we leverage spark's parallelism with GCS SDK ?			
createdTs, updatedTs, updateDate of Blob	Problem: GCS File properties are important metrics to know when a file is created and\or modified. Can we create functions to access those properties on GCS objects?			
isFile	Problem: GCS URI is hierarchical, many times developer is confused if a URI is file or a folder. Can we create a function to determine that?			
Audit package				

AuditRecord	Problem: For any streamlined & automated data pipeline, series of events happening during an execution at session level is important to audit. Example: 1. **Status** of an execution has cycle - INIT, STARTED, INPROGRESS, FAILED/SUCCEEDED so status is an important metric to audit 2. **Processing Time** or **Number of Rows processed** is another metric which can be audited from an data pipeline execution	Use case overview: Communicating between SQL & Scala needs an unified class. Case classes are beneficial for record level data. 1 session of pipeline execution creates 1 audit record	Overview: A SQL record auditing below listed attributes during execution of datapipeline `case class AuditRecord (yarnApplicationId: String, jobStartTime: Timestamp, jobEndTime: Option[Timestamp], runtimeMin: Option[Int], status: String, jobName: String, username: Option[String], refreshType: String, runDate: Timestamp, processedDates: Option[String], errorMsg: Option[String], lastPrcsdPartTs: Option[Timestamp], lastPrcsdRowTs: Option[Timestamp], batchId: Long)`	
initAuditRecord	Problem: When an Audit Record is initiated during execution of data pipeline few fields have default values or initial values. If number of attributes are more it creates dependency on developer to initialize those attributes with default values. Is it possible to jot down them together in a function?	Use case overview: A Data pipeline execution begins with event of capturing start time, status & other default values. A new AuditRecord is helpful to keep track of execution stages. This function helps to instantiate new record with default values & commits in audit table.	Overview: Function to initiate an audit record to capture series of events in a data pipeline.	

completedAuditRecord	Problem: Attributes in Audit Record need update values after complettion of execution with success or failure. If number of attributes are more it creates dependency on developer to initialize those attributes with default values. Is it possible to jot down them together in a function?	Use case overview: After completing the data refresh in table it is required to update the status of job, duration & processed rows. With this function all pre-defined set of attributes are updated after job's completion	Overview: Function to initiate an audit record to capture series of events in a data pipeline & commit into audit table	
insertAuditRecord	Problem: An AuditRecord is an instance of all attributes of Audit table with desired or default values. It's an overhead for a developer to write INSERT INTO query everytime. Plus the code in datapipeline become messy & code reviews or debugging becomes hard. Can we have a function to	Use case overview: When a data pipeline execution is initiated, an audit entry is required to track through all events happening duirng pipeline execution. This function will insert & commit initial values in Audit table	Overview: Function to insert, flush & commit an audit record into Audit table	
updateAuditRecord	Problem: Capturing all the events during a datapipeline run happening async need a function that updates one or many random audit table attributes..	Use case overview: When a data pipeline execution is underway, Audit record which is created is required to update on the events happening duirng pipeline execution. This function will update & commit updated values in Audit Table	Overview: Function to update, flush & commit an audit record into Audit table	
getPreviousRunAuditRecord	Problem: When a pipeline is initiated for execution before continuing it's important to know if the job run was restarted & fetch previously executed pipeline session details from audit table. Can we create a function to fetch previous executed AuditRecord?	Use case overview: When a data pipeline execution starts if it is a restarted job, Audit record which is created in previous job run is required. This function will fetch previous audit record & use that for restarted execution	Overview: Function to fetch audit details of previous batch run of a particular datapipleine	

getPrcsdRowTsFrmAuditRec	Problem: After a successful data refresh in hive table, can we have a function to know when was last row processed of the table?	Use case overview: Last row processed timestamp of a table is used in Debugging & Monitoring	Overview: Function to fetch last row processed created timestamp column value of a table	
overwriteAuditStatus	Problem: Status is critical attribute in audit table which has various values throughout the session during execution. Overwriting previous status value new & correct value needs a function to call when required.	Use case overview: After completing the data refresh in table it is required to update the status of job, duration & processed rows. With this function the status of job is overwriite with new value of Audit Record in Audit table	Overview: Function to overwrite previous value of status attribute in an audit record during execution of data pipeline.	
toDf	Problem: A list of audit records needs a convertor into dataframes to flush audit details into RDBMS database using spark	Use case overview: List of AuditRecords can be handled by wrapping into a RDD or Dataframe using spark for better experience	Overview: An implicit conversion of List of Audit Records to dataframe to operate using spark	
RDBMS Operations package				
getConnection	A function to fetch RDBMS connection details	Use case: Creates a connection to Audit database / tables, Supplier onboarding Database	Overview: Function to prepare RDBMS connection	
getDeleteStatement	Problem: As a Data Engineer, it's overhead to write DELETE SQL statements in a string because it may be error-prone and the end usage code can get messy with lot of delete conditions. We need an adaptable function to run through passed delete parameters & return DELETE SQL string	Use case: Prepares long string of Delete statement with conditions for RDBMS transactions	Overview: Function to prepare delete statement string	
addRecordsInDeleteStatement	A helper function to above function getDeleteStatement, adds a record with condition in Delete SQL	Use case: Helps to prepare delete statements with conditions on various columns	Overview: Function to add delete statement condition to delete statements	

getUpdateStatement	Problem: As a Data Engineer, it's overhead to write UPDATE SQL statements in a string because it may be error-prone and the end usage code can get messy with lot of update conditions. We need an adaptable function to run through passed update parameters & return UPDATE SQL string			
addRecordsInUpdateStatement	A helper function to above function getUpdateStatement, adds a record with condition in Update SQL			
getSelectStatement	Problem: As a Data Engineer, it's overhead to write SELECT SQL statements in a string because it may be error-prone and the end usage code can get messy with lot of SELECT clause SQL. We need an adaptable function to run through passed SELECT columns & return SELECT SQL string			
getInsertStatement	Problem: As a Data Engineer, it's overhead to write INSERT SQL statements in a string because it may be error-prone and the end usage code can get messy with lot of INSERT clause SQL. We need an adaptable function to run through passed INSERT columns & return INSERT SQL string			

addRecordsInInsertStatement	A helper function to above function getInsertStatement, adds a record with condition in INSERT SQL			
dbInsertOnly	Function to commit INSERT operation in RDBMS table			
processDbDelete	Function to commit DELETE operation in RDBMS table			
processDbUpdate	Function to commit UPDATE operation in RDBMS table			
processDbUpsert	Problem: Implement a MERGE operation for RDBMS. In case a primary key exists, UPDATE otherwise fresh INSERT			
Secret Manager package				
getSecret	Problem: Storing sensitive passwords & secret keys in codebase is discouraged. Need a utility to fetch passwords & secret keys stored in secret manager.	Use case: Passwords & Secret Access keys are critically sensitive so they are not put into codebase of Channel Performance We need a function to fetch secret access keys & password from Google Cloud's Secret Manager.	Overview: This is generic function that uses **gcloud CLI SDK** to fetc secrets from Secret Manager **USP of this function** is it uses bash command at CLI level to fetch secrets	
Common helpers package				
Email Email.send	Singleton object of Email having class variables for email Abstract send function to send an Email with subject & body, MIME types also supported	Use case: Email is used for communication & notification in Channel Performance for example Data Quality Rules execution Metrics, and other data pipeline related notifications	Overview: Uses well known javax email package, Independent case class object which has all bare minimum requirements for email. **USP is send function** that is implicit on Email object	
trySafely				
extractMatching	Function to extract regex matched substring from string			
isMatching	Function to identify if a string has matching regex substring			

systemExit	Function to call abrupt System Exit with EXIT CODE			
Data structure Converters				
HTTPS REST Module				
createHttpPostRequest				
createHttpClient				
createHttpClientWithRetries				
getRetryHandler				
getServiceUnavailableRetryStrategy				

- Data Quality- This class has methods to executeDQ, generateDQ reports which are used in our pipelines.
- ETL-
  - getAuditColumnsWithNewValues - This is to add new audit columns with new values- user_id, load_ts, upd_ts, delta.
  - prepareExecution - This is to create list of schemas to be created in the DB before pipeline execution
  - getSchema - This is to create the schema with a particular DDL.
  - performDelta- This method creates new column - "delta" on dataframe excluding primaryKeys by comparing the hash-code of columns in both source and stg dataframes and primaryKeys presence to find out Inserts , Updates, Deletes and No Changes.
  - backup- This method is used to compute backup at a location and returns boolean if all underlying files of external dataframe copied successful, also prints file-locations which did not succeed copy
  - cliParser- This method is used to parse the command line arguments for our data pipelines like refreshType, startDate, endDate etc.
- Table - This class holds methods for Hive Tables like
  - getTableLocation - which is used to fetch gcs location for any external table.
  - repairRefreshTable to refresh the external tables created in Hive.
  - drop and delete methods for dropping hive tables and deleting data in gcs.
  - Also has utilities like getAffectedPartitions & getAffectedBlobs which returns list of affected partitions including nested partition eg- for any source table it returns affected partitions after a particular LocalDateTime.

GCP

- ApiKey - This case class holds GCS Api key.
- GcsInterpolators- This class holds method which fetches Information of Blob for given string or creates or Fetches Blob Information of GCS Object
- GcsObject- This class has methods toGcsObject which Separates GCS URI String into organized parts viz. Bucket, Object/Blob (s)
- ObjectActions - This class has methods to copy , move gcs buckets, parallelCopy and parallelRemove , parallelMove for each partitions.

Audit

Audit utility is being used for following -

- Auditing metrics such as processed dates, duration of run, error if any etc.
- Capturing the timestamp till which data was read from source to identify affected partitions in next run
- Capturing the timestamp from which data is loaded to application table, used by feed generator to identify affected partitions
- Batch-wise auditing for a job

Utility Function	How does this utility helps in Data pipelines?	Flow / Design Diagram
initAuditRecord(baseAuditRecord: AuditRecord)	Creates a new audit record using an existing AuditRecord as a base. Typically used for duplicating or appending audit data.	Flow: Start Fetch Base Audit Record Create New Record with Modifications Store Record End
insertAuditRecord(auditRecord: AuditRecord, dbProps: Properties)	Inserts the given AuditRecord into the database, leveraging the provided database properties (dbProps). Used for logging workflow steps.	Flow: Start Validate Record Connect to Database Insert Record Log Status End
removeNanoFromTs(auditRecord: AuditRecord)	Removes nanoseconds from the timestamp in the given AuditRecord to ensure standardization of time values.	Flow: Start Extract Timestamp Truncate Nanoseconds Update Audit Record End
updateAuditRecord(auditRecord: AuditRecord, dbProps: Properties)	Updates an existing audit record in the database using the provided properties. Useful for modifying entries during a workflow execution.	Flow: Start Connect to Database Locate Record Update Fields Commit Changes End

getPreviousRunAuditRecord(jobName: String, dbMap: Map[String, String], refreshType: Array [String])	Fetches the previous run's audit record for the given job name and refresh type. Returns an Option [AuditRecord] for tracking execution history.	Flow: Start Fetch Job Name & Type Query Database for Previous Record Return Record (if exists) End
getPrcsdRowTsFrmAuditRec(jobName: String, dbMap: Map[String, String], refreshType: Array[String], lastRunTs: Timestamp)	Retrieves the timestamp of the last processed row from the audit record for a specific job. Helps ensure no duplication of data processing.	Flow: Start Fetch Audit Record Extract Last Processed Timestamp Compare with Input Timestamp Return End
overwriteAuditStatus(jobName: Array[String], refreshType: String, runDate: Timestamp, dbProps: Properties)	Overwrites the audit status for the given job name and refresh type on the specified run date. Useful for fixing job statuses after failures or re-runs.	Flow: Start Read Job Parameters Connect to Database Update Status for Jobs Commit Changes End
getJobAuditRecords(query: String, dbMap: Map[String, String])	Executes a query to fetch job audit records as a DataFrame for analysis and reporting.	Flow: Start Parse Query Connect to Database Execute Query Transform Result to DataFrame Return DataFrame End
implicit class DataframeFromAuditRecord (auditRecord: AuditRecord)	Provides an implicit conversion of an AuditRecord to a DataFrame. Useful for integrating with Spark-based workflows.	Flow: Start Extract Fields from Audit Record Map Fields to DataFrame Columns Return DataFrame End

CCM

CCM Utility is being used to fetch config from CCM

CcmUtils - This trait has following methods

- getCcmConfig - Returns CCM config for given config name, app name to be set before calling this method
- getCcmConfigAsProperties - Converts map returned by CCM to properties object
- getCcmProviderConfig - Returns CCM config for given config name and app name

Common

- Email - This class has function to send email from email address , to list of email address , email subject , email body , smtp, cc optional and bcc optional list.
- Helpers - This class has below methods
    - isMatching - Returns if match is succeeded for any given input with a regex provided.
    - extractMatching-Returns first matched substring from given input which matches any valid regular expression
    - trySafely - This is heavily generic higher-order function wrapper of Try/Catch block for embedding unsafe code & handle exceptions
    - systemExit - Exit code to exit system: Zero or Non-zero
- Convertors- This class has util methods to convert Java Array List to Scala List , Java Sets to Scala Sets etc , all types of java data structures to Scala.

Constants

This module will hold all static final constants used across projects.

Datetime

This utility has 2 main classes

- DateTimeHelpers- This is used to get dates between any two given dates which drives our history and restatements pipelines. Also has helper method to extract Date from a given string based on a given regex.
- DateTimeInterpolators - This utility helps to fetch date time for any given string based on multiple formats like year_moth_date, year_month_date_24hours etc.

HttpRest -

HttpRestOps has following methods-

- createHttpPostRequest - It is used to create http post request with headers and payload as entity.
- createHttpClient - It is used to create http client with retry handler and service unavailable strategy by calling createHttpClientWithRetries method. It helps us to make http call, even if it gets failed it keeps on retrying till the count becomes equal to retryThreshold and wait for exponential amount of time between each call. The below two methods are being used in its implementation-

1. getRetryHandler - It is used to create retry handler for http client based on retry threshold and interval as configured in application.properties.
2. getServiceUnavailableRetryStrategy - It is used to create service unavailable retry strategy implementation with retry threshold and retry count.

Schema -

Secrets

Secrets Utility is being used to fetch secret from Google Secret Manager

SecretOps has following method -

- getSecret - Returns secret for given project name, secret name and version

DatabaseOps -

ImplicitDatabaseOperation - It is used to perform the following operations using JDBC connection.

- dbInsertOnly - It is used to insert the records in the database.
- processDbDelete > Delete the records in the database. It will take three arguments which are database properties, delete conditions (it specifies the key on the basis of deletion takes place) and table name.
  > Prepare the delete statement and execute it in batches.
- processDbUpdate > Update the records in the database.
  > Take four arguments which are database properties, update conditions (it specifies the key on the basis of updation takes place), update columns (only these columns will change and other remains same) and table name.
  > Prepare the update statement and execute it in batches.
- processDBUpsert > Upsert the records in the database. If the record already exist in database, then update it, otherwise insert it.
  > Take five arguments which are database properties, select conditions, insert columns (columns which need to be inserted), update columns (columns which need to be updated), update conditions and table name.
  > Prepare three statements - select, insert and update. Furthermore, it is using **recordsExists** method to detect whether the record already exist in database or not.
  > If it exists, then add it in update batch, otherwise in insert batch and execute both the batches in last.