Efficient Strategies for Syncing via JDBC to Databricks Delta Lake

Synchronizing database records into Databricks Delta Lake can be a challenging task, especially when there is no straightforward way to read from a JDBC source as a stream.

This blog explores two effective approaches to sync data while ensuring that newly modified records are identified and can be processed or sent to an external system. We assume that transaction log-based Change Data Capture (CDC) tools like Oracle GoldenGate or AWS DMS are unavailable, either due to lack of access to the transaction log, because we are querying from a view or any other object that can't be data captured using a commit log.

Below, we outline two approaches to achieve this synchronization, along with considerations for reading the data, and options for running the pipeline continuously.

Considerations for Data Ingestion

Before diving into the approaches, it's crucial to assess the structure and characteristics of your source data. Key factors include:

1. **Primary Key Column**:
   - Does the source data have a primary key column?
   - If so, this can be used to merge records efficiently.
   - If not, the first approach (merging records) will not be applicable.
2. **Distribution Column**:
   - Is there a column that can be used to distribute records uniformly?
   - This column does not necessarily need to be unique but should allow for even distribution across partitions.

With these factors in mind, let's explore the two approaches:

Approach 1: Merging Records with a Last Updated Date Column

This approach is ideal when you have a primary key column in your source data. The strategy involves:

1. **Adding a Last Updated Date Column**:
   - Add a `last_updated_date` column to the source data if it doesn't already exist.
   - This column will be used to track when each record was last modified.

Code Snippet:

```
def get_table(source_table:str, target_table:str, ts_column:str):
    """
    This method downloads a table via jdbc into a stage Delta table using a table configuration that includes a
    partition column that can separate data without skew
    It first calculates the max value of such column to parallelize the retrieval, making the load much faster
    :param source_table: Source table name (JDBC)
    :param target_table: Target table name (Databricks)
    :param ts_column: a column on the delta table that represents the time the record was inserted, deleted, updated
    """
    (spark.read
        .format("jdbc")
        .option("url", f"jdbc:..........")
        .option("user", "username")
        .option("password", "password")
        .option("driver", "oracle.jdbc.driver.OracleDriver")
        .option("dbtable", source_table)
        .load()
        .withColumn(ts_column, F.now())
        .write
        .option("mergeSchema", "true")
        .format("delta")
        .mode("overwrite")
        .saveAsTable(target_table))

# First run
get_table("table_from_jdbc", "delta_table", "last_updated_date")
```

2. **Merge Logic**:
   - Use the `MERGE INTO` operation provided by Delta Lake to update existing records and insert new ones.
   - If a record exists in the Delta table with the same primary key, it is updated; otherwise, it is inserted as a new record.

Code Snippet:

```
def do_merge(table_to_update_name:str, stage_table_name:str, primary_keys:list, ts_column:str):

    """This function will 2 table names with the same column fields,
    it will also take a list of primary keys and will execute a merge on the first table.
    Finally, it will take a column name that represents a timestamp column in the source table
    that will take the value of the time when the record was updated.
    The merge statement has 3 conditions:
        - When matched: this mean that a record already exists with the given primary key(s),
        in this case it will update all columns from source table with data from source table.
        - When not matched by target: this mean that there are no existing records with the given primary key(s)
        in this case it will insert a new record
        - When not matched by source: this mean that there are is a record on target that is not present in source
        as it was probably deleted in this case it will delete the record on target

    Keyword arguments:
        :param table_to_update_name: current table, contains data from previous merge operations
        :param stage_table_name: new table, containing last data from the source system
        :param primary_keys: sequence of strings that represent primary key(s)
        :param ts_column: a symbolic column on the target table with the last time the row was updated
    """
    table_cols = [c["col_name"] for c in
                  spark.sql(f"describe table {table_to_update_name}").select("col_name").collect()]

    clean_cols = [c for c in table_cols if c not in [ts_column]]
    merge_stmnt = ", ".join([f"main.{key} = stage.{key}" for key in clean_cols])
    stmt_pt1 = [key for key in clean_cols] + [ts_column]

    insert_stmt_pt1 = ", ".join(stmt_pt1)
    insert_stmt_pt2 = ", ".join([f"stage.{c}" for c in clean_cols])

    update_check = " OR ".join([f"main.{key} != stage.{key}" for key in clean_cols])
    pk_stmnt = " AND ".join([f"main.{key} = stage.{key}" for key in primary_keys])

    merge = f"""
      MERGE INTO {table_to_update_name} AS main USING {stage_table_name} AS stage
      ON {pk_stmnt}
      WHEN MATCHED AND ({update_check}) THEN UPDATE SET {merge_stmnt}, main.{ts_column} = now()
      WHEN NOT MATCHED BY TARGET THEN INSERT ({insert_stmt_pt1}) VALUES ({insert_stmt_pt2}, now())
      WHEN NOT MATCHED BY SOURCE THEN DELETE
    """
    spark.sql(merge)

# New run
get_table("table_from_jdbc", "delta_table_stage", "last_updated_date")
do_merge("delta_table", "delta_table_stage", ["identifier_pk"], ts_column="last_updated_date")
```

Approach 2: Subtracting Dataframes

This approach is useful when you either lack a primary key column or prefer to work with the entire dataset in each run:

1. **Loading the Full Dataset**:
    - Load the full dataset from the source database into a DataFrame.

Code Snippet:

```
 # First run
get_table("table_from_jdbc", "delta_table", "last_updated_date")
```

2. **Subtracting Dataframes**:
    - Compare the new DataFrame with the previous one to identify newly added or modified records.
    - Use the `subtract` method to find the difference between the current and previous datasets.

Code Snippet:

```
def do_diff(table_to_update_name, stage_table_name, ts_column):
    """
    This method executes a diff operation to the target table using a staging or temporary table
     such table contains a copy of the latest snapshot of the source Oracle views
    A diff is a basic A - B operations that considers all rows with changes as new rows
    This approach can generate duplicates in the target table

     :param table_to_update_name: current table, contains data from previous merge operations
     :param stage_table_name: new table, containing last data from the source system
     :param ts_column: a symbolic column on the target table with the last time the row was updated
    """

    table_to_update = spark.read.table(table_to_update_name).drop(ts_column)
    stage_table = spark.read.table(stage_table_name).drop(ts_column)

    (
       stage_table
         .exceptAll(table_to_update)
         .withColumn(ts_column, F.expr("now()"))
         .write.mode("append").saveAsTable(table_to_update_name)
    )


# New run
get_table("table_from_jdbc", "delta_table_stage", "last_updated_date")
do_diff("delta_table", "delta_table_stage", "last_updated_date")
```

Conclusion

Synchronizing data from a database into Delta Lake in Databricks requires careful consideration of the data's structure and available columns. By using either a merge strategy or subtracting DataFrames, you can effectively manage and identify changes in your dataset, allowing you to keep your Delta table up-to-date and ready for further processing.

The choice of whether to run the pipeline in batch or continuous mode depends on your specific use case and performance requirements.