Implementing CDC and SCDType2 logic using Scalding

Author

Umesh Pawar
Associate - Projects
Cognizant Technology Solutions
Employee ID: 461757
Created on: September 27, 2015

Contents

Change data capture (CDC) to identify inserts, updates, deletes

Change Data Capture

A stage in the ETL processing where we compare today's file with yesterday's snapshot to identify changed records – Inserts, Updates and Deletes (deletes are applicable only in the cases where Source sends input file as a 'Full Snapshot')

CDC by Example

Yesterday's snapshot:

//id1, subid1, ccol1, ccol2, ccol3
//[---PK--------] [--------CDC Cols--------------]

("111", "AAA", "123", "text1", "moretext1")
("222", "AAA", "123", "text1", "moretext1")
("333", "CCC", "123", "text1", "moretext1") // Delete

Today's source file:

//id1, subid1, ccol1, ccol2, ccol3
//[---PK-------] [--------CDC Cols--------------]

("111", "AAA", "123", "text1", "moretext1") //Ignore as it's unchanged
("222", "AAA", "123", "text777", "moretext1") // Update to col 2
("444", "DDD", "123", "text1", "moretext1") // Insert

Output of CDC stage:

//id1, subid1, ccol1, ccol2, ccol3, cdc_flag
//[---PK---------] [----------CDC Cols-------------] [-FLAG-]

("222", "AAA", "123", "text777", "moretext1", "U") //Update
("333", "CCC", "123", "text1", "moretext1", "D") // Delete
("444", "DDD", "123", "text1", "moretext1", "I") // Insert

Pseudo code to implement CDC in Scalding:

Steps:

1. Read yesterday's snapshot data in pipe called snapshotPipe
2. Add indicator field with OLD value in snapshotPipe
3. Read today's source data in pipe called inputPipe
4. Add indicator field with NEW value in inputPipe
5. Make sure both pipe has similar schema
6. Combine both pipes in 3rd pipe called combinedPipe=(snapshotPipe + inputPipe)
7. The preprocessing of the combinedPipe should ensure that tuples are grouped by the **unique key** (*id1, subid1*) and sorted on the **indicator** so that NEW records come before OLD where both exist. Also the secondary sorting should be on **CDC columns** (all other columns except PK) so that all CDC cols with same values should occur in consecutive tuples in the window function **scanLeft** (within which this is called).
8. To identify modifications, we use the window function **foldLeft** on the records grouped by the **primary keys**.
    A. Within this method records are sorted (NEW -> OLD)
    B. If first record is OLD, it means no NEW records with key exist, hence record has been **DELETED.**
    C. If first record is NEW, it can be an insert, update or duplicate. We don't know before we see the next record. We mark them as INSERT. The next iteration of the window function will update, if necessary, as below:
        i. If left and right tuples are same, then we have a duplicate, so we emit blanks which are **discarded.**
        ii. If left and right tuples are different, the record is marked as **UPDATED.**
        iii. Anything else should not be the case, so we have a **RunTimeException** to catch unexpected output.

Slowly Changing Dimension

Dimensions in data management and data warehousing contain relatively static data about such entities as geographical locations, customers or products. A Slowly Changing Dimension (SCD) is a dimension that stores and manages both current and historical data over time in a data warehouse. Data captured by Slowly Changing Dimensions (SCDs) changes slowly but unpredictably, rather than according to a regular schedule.

It is considered and implemented as one of the most critical ETL tasks in tracking the history of dimension records.

The three types of SCDs are:

Type 1 SCD - Overwriting

In a Type 1 SCD the new data overwrites the existing data. Thus the existing data is lost as it is not stored anywhere else. Any additional information is not needed to specified to create a Type 1 SCD.

Type 2 SCD - Creating another dimension record

A Type 2 SCD retains the full history of values. When the value of a chosen attribute changes, the current record is closed. A new record is created with the changed data values and this new record becomes the current active record. Each record contains the effective time and expiration time to identify the time period between which the record was active.

Type 3 SCD - Creating a current value field

A Type 3 SCD stores two versions of values for certain selected level attributes. Each record stores the previous value and the current value of the selected attribute. When the value of any of the selected attributes changes, the current value is stored as the old value and the new value becomes the current value.

SCD2 logic using Scalding

SCD2 by Example:

Yesterday's snapshot:

[-PK-][-Data-][---eff_start_date---] [---eff_end_date------] [-------- load_tms-] [---updated_tms------]

("N1", "A1", "1", "2014-11-12 00:00:00.000", "9999-12-31 00:00:00.000", "2014-11-13 10:00:00.000", "2014-11-13 10:00:00.000")

("N2", "A2", "2", "2014-11-12 00:00:00.000", "9999-12-31 00:00:00.000", "2014-11-13 10:00:00.000", "2014-11-13 10:00:00.000")

Today's source data:

[-PK-][-Data-][---eff_start_date---] [---eff_end_date------] [-------- load_tms-] [---updated_tms------]

("N1", "A1_UPDATED", "1", "2014-11-14 00:10:30.000", "9999-12-31 00:00:00.000", "2014-11-15 20:00:00.000", "2014-11-15 20:00:00.000")

Output of SCDType2 phase:

[-PK-][-Data-][---eff_start_date---] [---eff_end_date------] [-------- load_tms-] [---updated_tms------]

("N2", "A2", "2", "2014-11-12 00:00:00.000", "9999-12-31 00:00:00.000", "2014-11-13 10:00:00.000", "2014-11-13 10:00:00.000"),

("N1", "A1_UPDATED", "1", "2014-11-14 00:10:30.000", "**9999-12-31 00:00:00.000**", "2014-11-15 20:00:00.000", "2014-11-15 20:00:00.000")

("N1", "A1", "1", "2014-11-12 00:00:00.000", **"2014-11-14 00:10:30.000**", "2014-11-13 10:00:00.000", "2014-11-15 20:00:00.000"))

Pseudo code to implement SCDType2 in Scalding:

Steps:

1. Read yesterday's snapshot data in pipe called snapshotPipe
2. Read today's source data in pipe called inputPipe
3. Add **'eff_start_date, 'eff_end_date,' load_tms, 'updated_tms** fields with **{[curr_bus_date_time," 9999-12-31 00:00:00.000", curr_bus_date_time, curr_bus_date_time]}** value respectively in inputPipe
4. Make sure both pipe has similar schema
5. Combine both pipes in 3rd pipe called combinedPipe=(snapshotPipe + inputPipe)
6. The preprocessing of the combinedPipe should ensure that tuples are grouped by the **unique key** (*PK*) and sorted on the **eff_start_date column** so that latest record come before OLD snapshot record where both exist.
7. Use **scanleft** function to swap the eff_end_date and updated_tms field in pipe.

a. In above example for A1 record, value of **'eff_start_date** field from new record which is current business date is assigned to 'eff_end_date of record present in old snapshot record. This process is called as logical delete of record from old snapshot file.

b. Code snippet:

```
combinedPipes
    .groupBy(surrogateSymbols) { group =>
      group.sortBy(sortSymbol)
        .reverse
        // The most recent is now on TOP row
        .scanLeft[(String, String, String, String), (String, String, String, String)]((startDate,
endDate, sortByKey, datawarehouseUpdatetmsField) -> (startDate, endDate, sortByKey,
datawarehouseUpdatetmsField))("", "", "", "") {
          (leftRecord, rightRecord) =>
            if (leftRecord == ("", "", "", "")) {
              rightRecord
            } else {
              (rightRecord._1, leftRecord._1, rightRecord._3, leftRecord._4)
            }
        }
      // remove the null entries (the side-effect of .scanLeft)
    }.filter(sortSymbol) { sortByField: String => sortByField != "" }
}
```