

Audit Balance and Control

Technical Specifications



August 17, 2021

Donyati India pvt. ltd.

# Change Log

|  |  |  |
| --- | --- | --- |
| Date | User | Changes |
| 17-Aug-2021 | Anirudh Pedaprolu | Initial Draft |
| 18-Aug-2021 | Naveen Saboji | Updated with Pre-Execution steps |

Contents

[Change Log 1](#_Toc80270037)

[Software requirements: 2](#_Toc80270038)

[Audit Pre-Execution: 2](#_Toc80270039)

[Components: 2](#_Toc80270040)

[1. Tables 2](#_Toc80270041)

[2. Stored Procedure: audit\_pre\_execution 2](#_Toc80270042)

[3. Talend Sample Code: 4](#_Toc80270043)

[Audit Post-Execution: 5](#_Toc80270044)

[Components: 5](#_Toc80270045)

[1. Tables 5](#_Toc80270046)

[2. Stored Procedure: audit\_post\_execution 5](#_Toc80270047)

[3. Talend Sample Code: 6](#_Toc80270048)

[Audit Error Logging 6](#_Toc80270049)

[Components: 6](#_Toc80270050)

[1. Tables 6](#_Toc80270051)

[2. Talend Sample Code: 7](#_Toc80270052)

[Reference Documents 8](#_Toc80270053)

# Software requirements:

**ETL tool:** Talend open Studio Big Data 7.3.1

**Database:** PostgreSQL

**Reporting Tool (Optional):** Power BI/ Tableau

**Note:** As part of POC the above tools were used.

# Audit Pre-Execution:

The pre-execution is triggered at the very beginning of the job execution. It is responsible to capture details of Job Details(Master/jobs), Audit trails for the each of the job being triggered and

## Components:

### Tables

* + 1. abc.job\_detail
    2. abc.master\_job\_detail
    3. abc.master\_run\_detail
    4. abc.job\_run\_detail
    5. abc.master\_subjob\_xref
    6. abc.job\_header\_summary

### Stored Procedure: audit\_pre\_execution

**Call Query:**

CALL ABC.audit\_pre\_execution(job\_name, master\_job\_name, job\_version, master\_job\_flag, appln\_name, src\_sys\_name, full\_load\_flag, master\_job\_trigger\_flag, run\_status\_flag);

The audit Pre-Execution procedure is used in 2 variations:

1. **Master Job Runs:**

In this variation the master\_job\_name, master\_job\_flag, Version\_id and run\_status\_flag for the job is sufficient. This procedure is triggered at the beginning of the master job run, It loads the below tables.

abc. master\_job\_detail

abc.master\_run\_detail

**Execution Steps:**

Step 1: Initiate Pre execution process

Step 2: Check if job entry exists in master\_job\_detail Once triggered If exists Continue with Step 4 else proceed with Step 3

Step 3: If No , Make an entry into master\_job\_detail with Below columns.

Master\_job\_name

version\_id

appln\_name

src\_sys\_name

created\_timestamp

created\_by

Step 4: For any individual jobs triggered Add an entry into the master\_run\_detail with below column values:

master\_run\_id

master\_job\_id

initiate\_timestamp

run\_status

full\_load\_flag

Step 5: End of the Pre execution process

1. **SubJob/ Individual Job Runs:**

In this variation the job\_name, master\_job\_flag, master\_job\_trigger\_flag, Version\_id and run\_status\_flag for the job is sufficient. This procedure is triggered at the beginning of the master job run, it loads the below tables.

abc. job\_detail

abc.job\_run\_detail

abc.master\_subjob\_xref

abc.job\_header\_summary

**Execution Steps:**

Step 1: Initiate Pre execution process

Step 2: Check if job entry exists in job\_detail Once triggered IF exists Continue with Step 4 else proceed with Step 3

Step 3: If No, Make an entry into job\_detail with Below columns.

job\_name

version\_id

appln\_name

src\_sys\_name

created\_timestamp

created\_by

Step 4: If subjob triggered from Master job then make an entry to master\_subjob\_xref if it’s already exists in master\_subjob\_xref otherwise continue with next step.

Step 5: For any individual jobs triggered Add an entry into the job\_run\_detail with below column values:

Run\_id

Master\_run\_ID

master\_run\_id

master\_job\_trigger\_flag

Job\_id

initiate\_time

run\_status

full\_load\_flag

Step 6: Check if the record exists in abc.job\_header\_summary If True Update the Existing record otherwise insert new record into the table with below column details.

job\_id

master\_job\_trigger\_flag

last\_triggered\_timestamp

last\_triggered\_run\_id

last\_status\_flag

src\_read\_count

tgt\_insert\_count

tgt\_update\_count

tgt\_delete\_count

unprocessed\_rec\_count

Step 7: End of the Pre execution process

### Talend Sample Code:

**Screenshot:**

Timeline

Description automatically generated with low confidence

**Component name**: tDBRow\_1\_sp\_audit\_post\_execution

# Audit Post-Execution:

The post-execution is triggered at the very end of the job execution. It is responsible to relay metrics, run status and execution time to the database storing the ABC data.

## Components:

### Tables

* + 1. abc.job\_run\_info
    2. abc.master\_run\_info
    3. abc.job\_header\_summary

### Stored Procedure: audit\_post\_execution

**Call Query:**

Call ABC.audit\_post\_execution(job\_run\_id, master\_run\_id, run\_status\_id, src\_read\_cnt, tgt\_ins\_cnt, tgt\_upd\_cnt, tgt\_del\_cnt, unproc\_cnt);

The audit post procedure is used in 2 variations:

1. **Master Job Runs:**

In this variation the job\_run\_id and metric information (src\_read\_cnt, tgt\_ins\_cnt, tgt\_upd\_cnt, tgt\_del\_cnt, unproc\_cnt) are not passed. The master\_run\_id and run\_status\_id is sufficient.

When this procedure is triggered at the end of the master job run, it passes the run status which is interpreted by the procedure at the database level.

**Execution steps:**

Step 1: initiate the Post execution process

Step 2: Determines the run status as ‘In Progress’, ‘Completed’ or ‘Failed’ using the run\_status\_id.

Step 3: Search the abc.master\_run\_detail table using the master\_run\_id for active i.e., run\_status=’In Progress’ job.

Step 4: Update the run status and completion\_timestamp with the system date and time.

With this in the ABC framework the master job run has finished.

Step 5: End the Post execution process

1. **SubJob/ Individual Job Runs:**

In this variation the job\_run\_id, run\_status\_id and metric information are passed. The metric information can be avoided in case of a job failure. The procedure defaults null metric values to 0 (zero).

**Execution steps:**

Step 1: initiate the Post execution process

Step 2: Determine the run status as ‘In Progress’, ‘Completed’ or ‘Failed’ using the run\_status\_id.

Step 3: Searche the abc.job\_run\_detail table using job\_run\_id for active i.e., run\_status= ‘In progress’.

Step 4: Update run status and completion\_timestamp with system date and time. Hence marking the run as finished in the ABC framework.

Step 5: In the case of run\_status=’Completed’ the make an entry to the abc.job\_metrics table. Using the job\_run\_id and metric information.

Step 6: Finally, irrespective of the run status, the procedure will also update the abc.job\_header\_summary table with:

1. If run\_status is ‘Completed’, then last\_completed\_timestamp, last\_completed\_run\_id, last\_status\_flag and metric information are updated using the job\_run\_id.
2. If run\_status is not ‘Completed’, then last\_execution\_timestamp and last\_status\_flag is updated using the job\_run\_id.

**Note:** In this case the metric information and last\_completed\_run still refer to the previous ‘Completed’ run.

Step 7: End the Post execution process

**Note:** The audit post procedure will raise an exception and stop execution if both master\_run\_id and job\_run\_id are passed together.

### Talend Sample Code:

**Screenshot:**

Calendar

Description automatically generated with medium confidence

**Component name**: tDBRow\_2\_ sp\_audit\_post\_execution

# Audit Error Logging

In the Talend environment the error logging part of the framework is triggered by the tLogCatcher component. There are two variations in capturing the errors.

## Components:

### Tables

* + 1. abc.job\_error\_info
    2. abc.master\_error\_info

1. **Master Job Runs:**

In the case the error is raised in a master job, then the information (master\_run\_id, error\_timestamp, error\_component, error\_description and error\_code) is written to abc.master\_error\_info table.

1. **SubJob/ Individual Job Runs:**

If the error is raised from a subjob or individual job, then the information (job\_run\_id, error\_timestamp, error\_component, error\_description and error\_code) is written to abc.job\_error\_info table.

**Execution Step:**

Step 1: Initiate Error capturing process

Step 2: Catch all log events raised by tWarn, tDie components and catch any java exceptions.

Step 3: The errors that have a priority less than or equal to 4 are filtered and only the higher priority errors are captured.

Step 4:Load all the errors into the respective error tables mentioned below.

abc.job\_error\_info

abc.master\_error\_info

Step 5: End Process

### Talend Sample Code:

Screenshot for Master jobs:

A picture containing timeline

Description automatically generated

Screenshot for Subjob:

Text

Description automatically generated with medium confidence

# Reference Documents

1. Audit framework tables [link](https://netorg573064.sharepoint.com/:x:/s/ETL_ABC_Framework/EawBEbosM_hJkNTsvXdwin0B8lVOpk0WBwxNYQdy7yahsA).
2. Data Model [link](https://netorg573064.sharepoint.com/:b:/s/ETL_ABC_Framework/EYK_0ebE6H5Dm5VPloY4NRcBPmgMPZYtQh_z2lpQs6tCEg?e=d8hLeF).