Bridging Oracle Integration Cloud and Database Efficiency

Table of Contents

Abstract: 3

Limitations: 3

Oracle Integration Cloud (OIC) has got certain limitations with respect to call the database. 3

Process Implemented: 4

Advantages of the Solution: 8

Abstract:

This blog provides a comprehensive guide to optimizing the integration between Oracle Integration Cloud (OIC) and the database.

Consider the following scenarios:

- **Long-running validation:** OIC needs to perform a database validation and transformation that takes more than 4 minutes. In such cases, the operation fails due to timeout limitations.
- **Validation for more than one source**: When the data volume exceeds 100,000 or 1 million records, load data from all sources into the staging tables and perform validations collectively.
- **Large data retrieval:** OIC needs to fetch data exceeding 10 MB in size from the database. This too results in failure due to payload size limitations.

In this blog post, we will explore effective strategies to overcome these challenges.
These approaches will help simplify and optimize the overall process.

Limitations:

Oracle Integration Cloud (OIC) has got certain limitations with respect to call the database.

1. **Database Adapters**
   (Includes Oracle Database Adapter, IBM DB2 Adapter, Microsoft SQL Server Adapter, MySQL Adapter, Oracle Autonomous Data Warehouse Adapter, Oracle Autonomous Transaction Processing Adapter, and Oracle Database Cloud Service Adapter)

Trigger Configurations:

a. *Polling Operations* support up to **10 MB** payload size with schema transformation.

Invoke Configurations:

b. *Stored Procedures, Table Operations, and Run PureSQL Statements* support up to **10 MB** payload size with schema transformation for all outbound operations.

2. Oracle Autonomous Data Warehouse Adapter, Oracle Autonomous Transaction Processing Adapter, Oracle Database Cloud Service Adapter, MySQL Adapter, Microsoft SQL Server Adapter, Oracle Database Adapter, and IBM DB2 Adapter

Starting with the **August 2021 release**, all new integrations that involve **stored procedure** or **PureSQL database operations** must complete within **240 seconds**. Operations exceeding this limit will result in a **timeout**.

3. Payload Limits for Connectivity Agent-Based Adapters

For structured payloads (JSON, XML):

a. SOAP and REST protocols: **Maximum payload size is** 50 MB**.**

For other protocols:

b. FTP and File adapters: **Maximum payload size is** 50 MB**.**

c. Database, JMS, MQ, Kafka, and similar adapters: **Maximum payload size is 10 MB.**

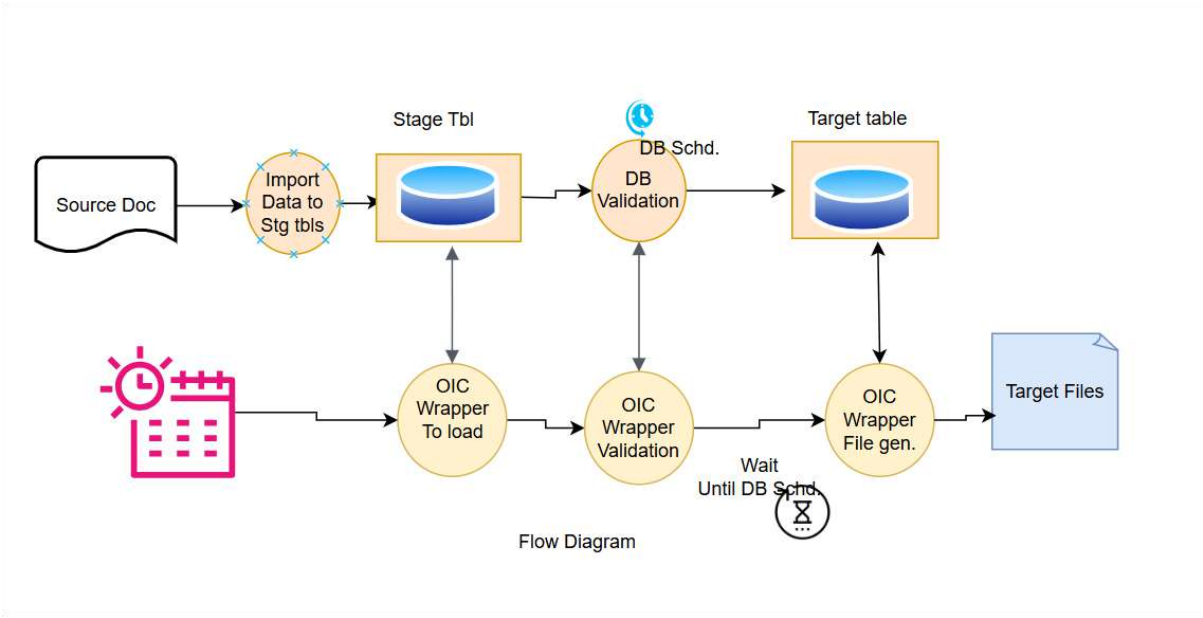
4.

Process Implemented:

To overcome the above limitations, the following approaches have been adopted:

1. **Utilizing the DB Scheduler** feature within the database.
2. **Implementing the Sub-Batching** concept to manage data efficiently.

The following process outlines the diagram for the same.



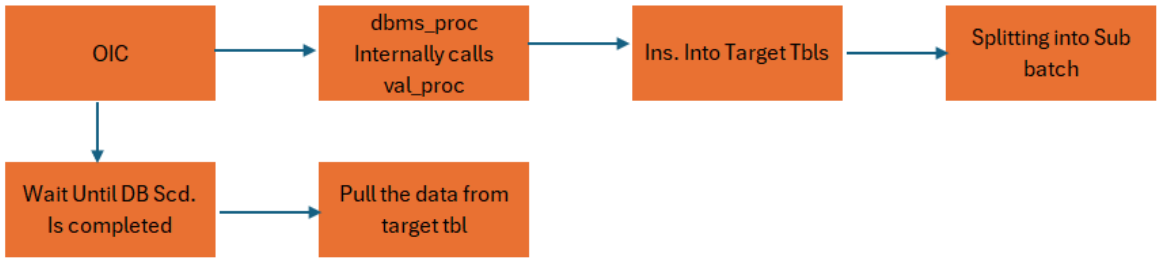
Solution:

To overcome the above limitations, a procedure has been implemented in the database to call the **DBMS_SCHEDULER**.

From Oracle Integration Cloud (OIC), a procedure (**XX_PKG.DBMS_PROC**) can be invoked, which internally creates a job (**CREATE_JOB**) using the scheduler to execute the actual validation and transformation procedure (**VAL_PROC**) and all other activities (splitting the data into sub batch, inserting data into target tables).

CREATE_JOB Argument Details:

```
DBMS_SCHEDULER.CREATE_JOB (
   job_name             IN VARCHAR2,
   job_type             IN VARCHAR2,
   job_action           IN VARCHAR2,
   number_of_arguments  IN PLS_INTEGER              DEFAULT 0,
   start_date           IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   repeat_interval      IN VARCHAR2                 DEFAULT NULL,
   end_date             IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   job_class            IN VARCHAR2                 DEFAULT 'DEFAULT_JOB_CLASS',
   enabled              IN BOOLEAN                  DEFAULT FALSE,
   auto_drop            IN BOOLEAN                  DEFAULT TRUE,
   comments             IN VARCHAR2                 DEFAULT NULL,
   credential_name      IN VARCHAR2                 DEFAULT NULL,
   destination_name     IN VARCHAR2                 DEFAULT NULL);
```



Flow Diagram#1

Code Snippet:

Note :In lc_annon_block above can be passed as parameter to call the actual validation process:

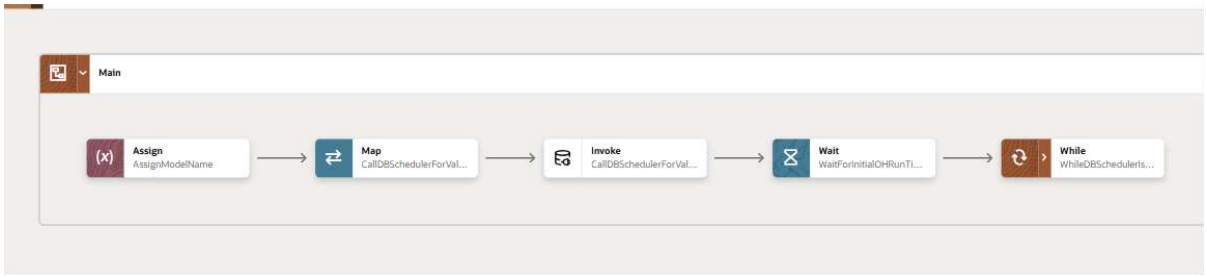
```
PROCEDURE dbms_proc (
    pp1 IN VARCHAR2
  , pp2 IN NUMBER
  , pp3 IN VARCHAR2
) IS

    lc_job_name   VARCHAR2(128) := p_model;
    lc_anon_block VARCHAR2(4000) := NULL;
    x_error_code  VARCHAR2(4000) := ' ';
    x_error_msg   VARCHAR2(4000) := ' ';
BEGIN
    dbms_output.put_line('Starting dbscheduler');
    lc_anon_block := 'BEGIN '
                  || 'XX_PKG.VAL_PROC'
                  || '('
                  || ''''
                  || pp1
                  || ''''
                  || ','
                  || ''''
                  || pp2
                  || ''''
                  || ','
                  || ''''
                  || pp3
                  || ''''
                  || '); '
                  || 'END;';

    dbms_output.put_line('lc_anon_block: ' || lc_anon_block);
    dbms_scheduler.create_job(job_name => 'A_'
                                       || lc_job_name
                                       || '_VT', job_type => 'PLSQL_BLOCK', job_action => lc_anon_block, enabled => TRUE, auto_drop => TRUE
                            , comments => 'Adhoc Job: ' || lc_job_name);

    COMMIT;
    dbms_output.put_line('Ending dbscheduler');
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line(sqlcode || sqlerrm);
END dbms_proc;
```

In this scenario, OIC will receive a **prompt** response from the database, as the procedure only submits the **scheduler** job, which internally triggers the **validation** and **transformation** process.



To determine whether the **DB Scheduler job** has completed, OIC can implement a check mechanism as outlined below, along with a configurable wait period. The check can be done through the below mentioned Oracle views.

Scheduler Views:

Oracle provides a set of views—**DBA_SCHEDULER_%, ALL_SCHEDULER_%**, and **USER_SCHEDULER_%**—to display detailed information about scheduler objects. These views are especially useful for monitoring and managing scheduled jobs.

1. DBA_SCHEDULER_JOB_ARGS
2. DBA_SCHEDULER_JOB_CLASSES
3. DBA_SCHEDULER_JOB_LOG
4. **DBA_SCHEDULER_JOB_RUN_DETAILS**
5. **DBA_SCHEDULER_JOBS**
6. DBA_SCHEDULER_PROGRAM_ARGS
7. DBA_SCHEDULER_PROGRAMS
8. DBA_SCHEDULER_RUNNING_JOBS
9. DBA_SCHEDULER_SCHEDULES
10. DBA_SCHEDULER_WINDOW_DETAILS
11. DBA_SCHEDULER_WINDOW_GROUPS
12. DBA_SCHEDULER_WINDOW_LOG
13. DBA_SCHEDULER_WINDOWS
14. DBA_SCHEDULER_WINGROUP_MEMBERS

Among these, the **DBA_SCHEDULER_JOB_RUN_DETAILS** view is particularly valuable. It provides a complete history of job executions, including run statuses and any error messages generated during failed runs.

```
1  select job_name,log_date,status,actual_start_date,run_duration from dba_scheduler_job_run_details where owner = [redacted]
2  and job_name like '%ONHAND%';
3
4  select * from dba_scheduler_jobs where owner [redacted];
```

Query Result

SQL | All Rows Fetched: 21 in 0.068 seconds

	JOB_NAME	LOG_DATE	STATUS	ACTUAL_START_DATE	RUN_DURATION
1	A[redacted]_VAL_DATA1	22-JUN-25 08.59.29.416679000 AM GMT	SUCCEEDED	22-JUN-25 08.49.35.315448000 AM ETC/UTC	+00 00:09:54.000000
2	A[redacted]_VAL_DATA1	21-JUN-25 12.37.36.945752000 PM GMT	SUCCEEDED	21-JUN-25 12.27.51.282122000 PM ETC/UTC	+00 00:09:46.000000
3	A[redacted]_VAL_DATA2	22-JUN-25 08.58.40.411123000 AM GMT	SUCCEEDED	22-JUN-25 08.49.34.089148000 AM ETC/UTC	+00 00:09:06.000000
4	A[redacted]_VAL_DATA2	21-JUN-25 12.37.31.283464000 PM GMT	SUCCEEDED	21-JUN-25 12.27.49.673668000 PM ETC/UTC	+00 00:09:42.000000
5	A[redacted]_VAL_DATA3	22-JUN-25 08.59.00.499414000 AM GMT	SUCCEEDED	22-JUN-25 08.49.35.317380000 AM ETC/UTC	+00 00:09:25.000000
6	A[redacted]_VAL_DATA3	21-JUN-25 12.36.39.121793000 PM GMT	SUCCEEDED	21-JUN-25 12.27.51.277192000 PM ETC/UTC	+00 00:08:48.000000
7	A[redacted]_VAL_DATA4	22-JUN-25 08.58.03.220829000 AM GMT	SUCCEEDED	22-JUN-25 08.49.34.300540000 AM ETC/UTC	+00 00:08:29.000000
8	A[redacted]_VAL_DATA4	21-JUN-25 12.37.41.104736000 PM GMT	SUCCEEDED	21-JUN-25 12.27.49.965342000 PM ETC/UTC	+00 00:09:51.000000
9	A[redacted]_VAL_DATA5	22-JUN-25 08.59.09.502503000 AM GMT	SUCCEEDED	22-JUN-25 08.49.35.509962000 AM ETC/UTC	+00 00:09:34.000000

In case it gets stuck, then by below command it can be drooped:

DBMS_SCHEDULER.drop_job('MY_TEST_JOB');

For Sub batching we can update sub_batch_id column as below:

Code Snippet:

```
udpate tbl set sub_batch_id = ceil(rownum/5000) where batch_id = #p_batch_id
```

Advantages of the Solution:

1. Prevents errors during OIC execution.
2. Consolidates all database operations into a single call for multiple data source files, reducing database interactions. (prevents time out issue within 4 mins.)
3. Utilizes sub-batching to keep each payload below 10 MB, enabling efficient data retrieval.

About Author: -

Samir has over over 14 years of IT experience, currently working at Trinamix Systems Pvt. Ltd since Sep'23. He has worked at LTI Mindtree since (Feb,22 to Aug23) and TCS from March 2010 to February 2022. His role includes as a Solution Architect (Techno-Functional Consultant) specialising in Fusion Cloud - SCM/Finance, I excel in various domains such as Business Requirement Gathering, Gap & Risk Analysis, System Design, Application Software Development, Implementation, and Application Testing.