

Certainly! Let's delve into a detailed explanation of the PL/SQL procedure `GE_PLN_REQUEST_SET`. This procedure is designed to manage and execute jobs based on conditions defined in a mapping table and status checks in another database table. Let's break down its components and functionality step by step:

1. Procedure Header and Parameters

```
sql
Copy code
PROCEDURE GE_PLN_REQUEST_SET(
    p_activity_name      IN  VARCHAR2,
    P_ACTIVITY_TYPE      IN  VARCHAR2,
    x_status             OUT VARCHAR2,
    x_message            OUT VARCHAR2
) IS
```

- **Parameters:**

- `p_activity_name`: Input parameter representing the name of the activity.
- `P_ACTIVITY_TYPE`: Input parameter representing the type of the activity.
- `x_status, x_message`: Output parameters to return status and messages from the procedure.

2. Constants and Variables Initialization

```
sql
Copy code
C_MAPVALUE          VARCHAR2(100) := 'DAILY';
C_EXECUTE           VARCHAR2(100) := 'N';
C_NUMBER_OF_RUN     NUMBER := 0;
V_DAY              VARCHAR2(100) := 'FRI';
V_SQL              CLOB := 'NA';
V_WAIT_TIME        NUMBER := 0;
V_STATUS           NUMBER := 0;
PARENT_activity_name VARCHAR2(1000);
```

- **Constants and Variables:**

- `C_MAPVALUE, C_EXECUTE`: Constants initialized to default values.
- `C_NUMBER_OF_RUN`: Variable to hold the count of runs.
- `V_DAY`: Variable to store the current day of the week.
- `V_SQL`: CLOB variable to store dynamic SQL statements.
- `V_WAIT_TIME, V_STATUS`: Variables used for wait time and status checks.
- `PARENT_activity_name`: Variable to store the parent activity name.

3. Cursor Declaration

```
sql
Copy code
CURSOR C_REQUEST_SET(PARENT_activity_name VARCHAR2, P_ACTIVITY_TYPE VARCHAR2) IS
    SELECT *
    FROM PDS.GE_PLN_FUNCTIONAL_MAPPING_TBL
    WHERE MAPPING_TYPE = 'REQUESTSET'
        AND INTERNAL_USE = PARENT_activity_name
        AND ACTIVITY_TYPE = P_ACTIVITY_TYPE
        AND CHAR_VALUE1 = 'Y'
    ORDER BY MAP_VALUE1, NUMERIC_VALUE1;
```

- **Cursor:**

- C_REQUEST_SET is a cursor that selects records from the GE_PLN_FUNCTIONAL_MAPPING_TBL based on certain criteria related to activity mappings (MAPPING_TYPE, INTERNAL_USE, ACTIVITY_TYPE, CHAR_VALUE1).

4. Main Execution Block

```
sql
Copy code
BEGIN
    SELECT TO_CHAR(sysdate, 'DY') INTO V_DAY FROM dual;
    PARENT_activity_name := p_activity_name;

    FOR C_REQUEST_SET_REC IN C_REQUEST_SET(PARENT_activity_name, P_ACTIVITY_TYPE) LOOP
        -- Loop through each record fetched by the cursor
```

- **Initialization:**

- Fetch the current day (V_DAY) from the database.
- Assign the input parameter p_activity_name to PARENT_activity_name.

5. Conditional Execution Based on Mapping Values

```
sql
Copy code
    C_MAPVALUE := NVL(C_REQUEST_SET_REC.MAP_VALUE1, 'DAILY');
    C_EXECUTE := 'N';

    IF (C_MAPVALUE = 'DAILY' AND C_REQUEST_SET_REC.NUMERIC_VALUE2 IS NULL) THEN
        C_EXECUTE := 'Y';
    ELSIF (C_MAPVALUE = 'DAILY' AND C_REQUEST_SET_REC.NUMERIC_VALUE2 = 1) THEN
        -- Check if daily execution with a limit
    ELSIF (C_MAPVALUE = 'WEEKLY' AND NVL(C_REQUEST_SET_REC.MAP_VALUE2, 'FRI') = V_DAY)
THEN
        -- Check if weekly execution matching current day
    ELSIF (C_MAPVALUE = 'MONTHLY') THEN
        -- Check if monthly execution based on current month
    END IF;
```

- **Conditional Checks:**

- Depending on MAP_VALUE1 (C_MAPVALUE), determine whether to set C_EXECUTE to 'Y' (execute) or 'N' (do not execute).
- Different conditions (DAILY, WEEKLY, MONTHLY) are evaluated based on the values in the mapping table (GE_PLN_FUNCTIONAL_MAPPING_TBL).

6. Job Execution Logic

```
sql
Copy code
    IF (C_EXECUTE = 'Y') THEN
        -- Construct SQL to execute the transformation flow
        V_SQL := 'declare x_status varchar2(1000); x_message varchar2(1000); begin
GE_PLN_TRANSFORMATION_CALL.GE_PLN_TRANSFORMATION_FLOW('' ' ||
                C_REQUEST_SET_REC.ACTIVITY_NAME ||
                ''',x_status,x_message); end;';

    BEGIN
        -- Create a job using DBMS_SCHEDULER
        DBMS_SCHEDULER.create_job (
```

```

        job_name          => 'SPM_CHILD_JOB' || C_REQUEST_SET_REC.ACTIVITY_NAME,
        job_type          => 'PLSQL_BLOCK',
        job_action        => V_SQL,
        start_date        => SYSTIMESTAMP,
        repeat_interval   => 'freq=hourly; byminute=0',
        end_date          => NULL,
        enabled           => TRUE,
        comments          => 'SPM_CHILD_JOB'
    );

    -- Run the created job immediately
    DBMS_SCHEDULER.run_job (
        job_name          => 'SPM_CHILD_JOB' || C_REQUEST_SET_REC.ACTIVITY_NAME,
        use_current_session => TRUE
    );

EXCEPTION
    WHEN OTHERS THEN
        -- Handle exceptions if job creation or execution fails
        x_status := 'E';
        x_message := 'ERROR';
END;
```

- **Job Creation and Execution:**

- If C_EXECUTE is 'Y', construct a PL/SQL block (V_SQL) dynamically to execute a transformation flow (GE_PLN_TRANSFORMATION_FLOW) for the current ACTIVITY_NAME.
- Use DBMS_SCHEDULER to create a job (SPM_CHILD_JOB) with specific parameters (job_name, job_action, repeat_interval).
- Immediately run the job and handle any exceptions that might occur during job creation or execution.

7. Wait Logic for Job Completion

sql

Copy code

```

-- Wait for job completion logic
v_wait_time := 0;
v_status := 0;

WHILE v_wait_time = 0 LOOP
    BEGIN
        -- Check status of the process
        SELECT COUNT(*)
        INTO v_status
        FROM GEMS_IFACE_SPM_TABLE
        WHERE ACTIVITY_NAME = C_REQUEST_SET_REC.ACTIVITY_NAME
        AND CREATION_DATE >= SYSDATE - 720 / 1440
        AND PROCESS_ID = (
            SELECT PROCESS_ID
            FROM (
                SELECT PROCESS_ID
                FROM GEMS_IFACE_SPM_TABLE
                WHERE ACTIVITY_NAME = C_REQUEST_SET_REC.ACTIVITY_NAME
                AND CREATION_DATE >= SYSDATE - 720 / 1440
                AND NVL(STATUS_FLAG, 'I') NOT IN ('E', 'C')
                ORDER BY CREATION_DATE DESC
            )
            WHERE ROWNUM = 1
        )
        AND NVL(STATUS_FLAG, 'I') NOT IN ('E', 'C');
```

```

EXCEPTION
    WHEN OTHERS THEN
        v_status := 0;
END;

IF (v_status = 0) THEN
    v_wait_time := 1;
ELSE
    BEGIN
        -- Wait for a certain period before checking again
        DBMS_SESSION.sleep(30);
    EXCEPTION
        WHEN OTHERS THEN
            -- Handle any exceptions during wait time
            x_status := 'E';
    END;
END IF;
END LOOP;

```

- **Wait Logic:**

- Initialize v_wait_time and v_status.
- Enter a loop (WHILE) to continuously check the status of the process until completion.
- Within the loop, query GEMS_IFACE_SPM_TABLE to determine if the process (PROCESS_ID) has completed.
- If the process is ongoing (v_status = 0), wait for a specified time (DBMS_SESSION.sleep(30)).
- Handle exceptions that may occur during the wait period.

8. Job Cleanup

```

sql
Copy code
    -- Drop the job after completion
    DBMS_SCHEDULER.drop_job (job_name =>
'SPM_CHILD_JOB' || C_REQUEST_SET_REC.ACTIVITY_NAME);

```

- **Job Cleanup:**

- Once the job's execution and wait conditions are satisfied, drop (DBMS_SCHEDULER.drop_job) the job (SPM_CHILD_JOB) associated with the current ACTIVITY_NAME.

9. Error Handling

```

sql
Copy code
    EXCEPTION
    WHEN OTHERS THEN
        -- Handle any unexpected errors
        x_status := 'E';
        x_message := 'ERROR';

```

- **Error Handling:**

- Throughout the procedure, exceptions (WHEN OTHERS) are caught to set the status (x_status) and message (x_message) in case of errors.

10. Loop Completion

```
sql
Copy code
    END LOOP; -- End of cursor loop
END GE_PLN_REQUEST_SET;
```

- **Loop Completion:**

- End of the FOR loop iterating over C_REQUEST_SET_REC, which processes each record from GE_PLN_FUNCTIONAL_MAPPING_TBL.

Summary

The GE_PLN_REQUEST_SET procedure dynamically **manages job execution** based on **conditions** specified in a **mapping table** (GE_PLN_FUNCTIONAL_MAPPING_TBL) and **checks the status** of associated **processes** in GEMS_IFACE_SPM_TABLE. It uses DBMS_SCHEDULER to **create, execute, and monitor jobs** (SPM_CHILD_JOB) **for each activity** defined by ACTIVITY_NAME. The procedure **handles** various **conditions** (DAILY, WEEKLY, MONTHLY) for **job execution**, waits for job completion, and **cleans up resources** after job completion or failure, ensuring robust execution and error handling throughout the process.

Sure, let's delve deeper into the PL/SQL code in the file "ge transformation call.sql":

1. **Package Body:** The code is a PL/SQL package body for the GE_PLN_TRANSFORMATION_CALL package. This package is a collection of procedures and cursors that handle data transformation flows related to GE_SPM_RULE_HEADERS_ALL.
2. **Main Procedure - GE_PLN_TRANSFORMATION_FLOW:** This is the main procedure that handles the transformation flow for a given activity name. It updates the status and messages throughout the process, executes transformations for data streams, handles errors, and updates records. It also includes calls to other stub procedures and updates to SPM details.
3. **Cursors:** The package body includes cursors for fetching data streams. Cursors are database objects used to retrieve rows from a result set one at a time.
4. **Debug Mode:** There is conditional logic for enabling debug mode. Debug mode is typically used during development to help identify issues or bugs in the code.
5. **Additional Procedure - GE_PLN_REQUEST_SET:** This procedure schedules and executes jobs based on daily, weekly, or monthly intervals. It ensures that jobs are not run more than once within the specified time frame and waits for the completion of the master transformation logic before proceeding.
6. **Error Management:** Exception handling is used to capture and log errors during the execution of these procedures. This is crucial for maintaining the integrity of the data and the smooth operation of the data transformation processes.
7. **Database Interactions:** The script includes updating and inserting into a database table named GEMS_IFACE_SPM_TABLE, calling other procedures and functions (like

GE_IFACE_SPM_DETAILS.GE_IFACE_SPM_UPDATE_END), and scheduling jobs using DBMS_SCHEDULER.

8. **Job Scheduling:** The script includes logic to handle daily, weekly, and monthly job execution based on certain conditions, and it has a mechanism to wait for the completion of certain processes before proceeding.

Overall, the code is designed to manage data flow and transformation processes within an Oracle database environment, with extensive use of exception handling to manage errors during the execution of these processes. It's a well-structured piece of code that takes into account various scenarios and conditions for data transformation. I hope this detailed analysis helps! If you have any other questions, feel free to ask. 😊

Sure, I'd be happy to explain the logic behind the GE_PLN_REQUEST_SET procedure in more detail. This procedure is invoked within the main procedure GE_PLN_TRANSFORMATION_FLOW.

Here's a breakdown of its functionality:

Invocation: The GE_PLN_REQUEST_SET procedure is called with three parameters: p_activity_name, 'BEFORE_IP', and two variables x_status and x_message which are used to capture the status and any messages generated by the procedure.

Purpose: The purpose of this procedure is to handle specific transformation tasks before the main transformation flow begins. The 'BEFORE_IP' parameter suggests that this procedure is handling tasks that need to be completed before the 'Inbound to Processing' (INBD to PRSD) flow starts.

Status Update: After the GE_PLN_REQUEST_SET procedure is called, the status of the procedure (p_status) is updated with the value of x_status. If x_status is NULL, then p_status is set to 'S' (presumably for 'Success').

Message Generation: A message is generated indicating that the stub execution for the INBD to PRSD flow has started for the given activity name. This message is stored in the x_message variable.

Debug Mode: If debug mode is enabled (v_debug_status = 'YES'), a debug message is generated and stored in the v_debug_message variable.

SPM Details Update: The GE_IFACE_SPM_DETAILS.GE_IFACE_SPM_UPDATE procedure is called to update the SPM details with the generated message and debug message.

Error Handling: If any errors occur during the execution of these steps, they are captured in the exception handling block, and the status and message variables are updated accordingly.

In summary, the GE_PLN_REQUEST_SET procedure appears to be a crucial part of the data transformation process, handling tasks that need to be completed before the main

transformation flow begins, updating status and messages, and managing errors. I hope this explanation helps! If you have any other questions, feel free to ask. 😊

In the PL/SQL code you provided, the debug mode is controlled by a value in the GE_PLN_SYSTEM_CONTROLS table. Here's how it works:

A SELECT query is run on the GE_PLN_SYSTEM_CONTROLS table to fetch the value of the CONTROL_TYPE field where CONTROL_TYPE is 'DEBUG_CONTROL' and ACTIVITY is the activity name (p_activity_name).

This value is stored in the v_debug_status variable.

If the SELECT query doesn't return a value or encounters an error, an exception is raised and v_debug_status is set to 'NO', effectively disabling debug mode.

If v_debug_status is not 'YES', a message is generated stating that debug mode is disabled for the activity. If v_debug_status is 'YES', a message is generated stating that debug mode is enabled for the activity.

Here's the relevant code snippet:

```
BEGIN

SELECT VALUE

  INTO v_debug_status

  FROM GE_PLN_SYSTEM_CONTROLS

  WHERE CONTROL_TYPE = 'DEBUG_CONTROL'

  AND ACTIVITY = p_activity_name;

EXCEPTION

  when others then

    v_debug_status := 'NO';

END;

IF (v_debug_status != 'YES') THEN

  v_debug_message := 'Debug mode is disabled for the ' ||

    p_activity_name;
```

ELSE

```
v_debug_message := 'Debug mode is enabled for the ' ||
```

```
p_activity_name;
```

END IF;

So, the debug mode is enabled or disabled based on the value in the GE_PLN_SYSTEM_CONTROLS table for the specific activity. I hope this explanation helps! If you have any other questions, feel free to ask. 😊

Sure, I'd be happy to explain the logic behind the GE_PLN_TRANSFORMATION_FLOW procedure in more detail. This procedure is the main procedure in the PL/SQL package body for the GE_PLN_TRANSFORMATION_CALL package.

Here's a breakdown of its functionality:

Initialization: At the start of the procedure, several variables are initialized, including p_status, p_start_date, and p_sequence. The p_status variable is set to 'S' (presumably for 'Success'), p_start_date is set to the current system date, and p_sequence is set to the next value of the IFACE_SPM_TABLE_S sequence.

Debug Mode: The procedure checks the GE_PLN_SYSTEM_CONTROLS table to determine if debug mode is enabled for the activity. If debug mode is enabled (v_debug_status = 'YES'), a debug message is generated.

SPM Details Insertion: The GE_IFACE_SPM_DETAILS.GE_IFACE_SPM_INSERT procedure is called to insert a new record into the GE_IFACE_SPM_DETAILS table with the status and debug message.

Transformation Flow: The procedure then enters a loop where it fetches data streams from the GE_SPM_RULE_HEADERS_ALL table and performs transformations on them. The transformations are performed in order of the DATASTREAM_WEIGHTAGE field.

Error Handling: If any errors occur during the execution of these steps, they are captured in the exception handling block, and the status and message variables are updated accordingly.

In summary, the GE_PLN_TRANSFORMATION_FLOW procedure orchestrates the data transformation process, managing the flow of data, updating status and messages, and handling errors. It's a crucial part of the data transformation process in the GE_PLN_TRANSFORMATION_CALL package. I hope this explanation helps! If you have any other questions, feel free to ask. 😊