GE\_PLN\_REQUEST\_SET

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 executio**n 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.

ge transformation call

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. 😊

GE\_PLN\_REQUEST\_SET procedure in more detail

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. 😊

Debug mode

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.

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. 😊

GE\_PLN\_TRANSFORMATION\_FLOW procedure in more detail

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. 😊

gems\_po.gehc\_po\_alloc\_req\_iface\_pkg.gehc\_po\_alloc\_rec\_req\_prc

Certainly! Let’s summarize the PL/SQL code from the uploaded file. Here are the key points:

1. **Procedure Name**: The code defines a procedure named gehc\_po\_alloc\_rec\_req\_prc.
2. **Purpose**: This procedure appears to handle allocation recommendations for purchase orders.
3. **Functionality**:
   * It checks whether a p\_load\_id (load identifier) is provided. If not, it sends an email notification and logs an error.
   * If a valid p\_load\_id is provided, it updates records in the gehc\_svc\_pln\_spm\_stg table.
   * It calls validation procedures (gehc\_bo\_vs\_alloc\_quantity and gehc\_allocation\_valid\_prc) based on certain conditions.
   * Finally, it handles exceptions and sets appropriate error flags.

Remember, this is a high-level summary, and the actual code may contain additional details and logic. If you need further specifics, feel free to ask!

Certainly! Let’s break down the PL/SQL code snippet you provided. Here are the key points:

* **Procedure Name**: gehc\_po\_alloc\_rec\_req\_prc
  + This procedure processes allocation recommendations for purchase orders.
  + It takes three parameters: p\_errbuf (OUT), p\_retcode (OUT), and p\_load\_id (IN).
* **Error Handling**:
  + If the p\_load\_id is null, it sends an email notification and logs an error.
  + Otherwise, it updates records in the staging table based on certain conditions.
* **Validation and Processing**:
  + Validates interface enable flag and calls validation procedures.
  + Handles backorder prioritization if a specific profile option is set.
  + Calls procedures to submit requisitions for valid records.
* **Exception Handling**:
  + Properly handles exceptions, logging errors and setting flags.

Remember, this is just a concise summary. The actual code may contain additional details and logic. If you need further clarification, feel free to ask! 😊

Certainly! Let’s dive into the validation procedures within the PL/SQL code snippet you provided. These procedures play a crucial role in ensuring data integrity and correctness during the processing of allocation recommendations for purchase orders. Here’s what’s happening:

1. **Interface Enable Flag Validation**:
   * The code checks whether the interface is enabled. This involves querying a value from the GEMS\_GLP\_INTERFACE\_VALUESET value set. If the interface is not enabled, an error message is generated.
   * Essentially, this step ensures that the necessary pre-requisites are in place before proceeding with further processing.
2. **Backorder Quantity Validation (Optional)**:
   * If a specific profile option (SPM\_PRIORITIZE\_BACKORDER) is set to ‘Y’, the procedure calls the gehc\_bo\_vs\_alloc\_quantity validation procedure.
   * This validation likely compares backorder quantities against allocation quantities to ensure that allocations are prioritized correctly.
3. **Allocation Validations**:
   * The procedure then calls the gehc\_allocation\_valid\_prc procedure. This likely performs additional checks on the allocation data, such as verifying that required fields are populated correctly.
   * Any errors encountered during this validation are handled appropriately.
4. **Requisition Submission**:
   * If valid records exist (i.e., the count of valid records is greater than zero), the procedure calls the gehc\_allocation\_reqimp\_prc procedure.
   * This step likely involves submitting requisitions based on the allocation recommendations.
5. **Error Handling**:
   * Throughout these steps, the code handles exceptions and logs relevant error messages.
   * The l\_error\_flag is set to ‘Y’ if any errors occur, and the l\_err\_code and l\_error\_msg capture details about the error.

Remember, this is a high-level overview, and the actual validation procedures may involve more intricate logic. If you have specific questions about any of these steps or need further clarification, feel free to ask! 😊

Certainly! Let’s explore how requisitions are submitted in the allocation recommendation process outlined in the PL/SQL code snippet you provided.

1. **Validation and Preparation**:
   * Before submitting requisitions, the code performs various validations and checks. These include ensuring that the interface is enabled, validating backorder quantities (if applicable), and checking the validity of allocation data.
   * If any errors or discrepancies are found during these validations, appropriate error handling occurs, and the process may be halted.
2. **Requisition Submission**:
   * Assuming all validations pass, the procedure proceeds to submit requisitions. Here’s how it typically works:
     + The code queries the staging table (gehc\_svc\_pln\_spm\_stg) for valid records related to allocation recommendations.
     + If valid records exist (i.e., the count of valid records is greater than zero), the procedure calls the gehc\_allocation\_reqimp\_prc procedure.
     + Inside gehc\_allocation\_reqimp\_prc, the actual requisition submission logic takes place. This likely involves creating requisition records in the appropriate Oracle applications tables (such as PO\_REQUISITIONS\_INTERFACE\_ALL).
     + The requisition data (e.g., item details, quantities, delivery information) is prepared based on the allocation recommendations.
     + The requisitions are then submitted to the relevant Oracle purchasing module (e.g., Purchasing, iProcurement) for further processing.
3. **Error Handling and Logging**:
   * Throughout this process, the code handles exceptions and logs relevant information. If any errors occur during requisition submission, they are captured in the error\_message column of the staging table.
   * Additionally, the gems\_iface\_process\_table records details about the process, including start and end times, status, and any relevant messages.

Remember, this is a simplified overview, and the actual implementation may involve additional steps specific to your organization’s business rules and Oracle setup. If you have further questions or need more details, feel free to ask! 😊

GEHC\_SVC\_PLN\_FEEDBACK\_PRG\_PKG

Certainly! Let’s dive into the details of the PL/SQL code from the uploaded file. Here are the key points:

1. **Package Purpose**:
   * The package is named GEHC\_SVC\_PLN\_FEEDBACK\_PRG\_PKG.
   * It handles feedback recommendations for planning purposes and purges records from a staging table.
   * Feedback types include “Feedback” and “Purge.”
2. **Extracting PO and SO Details**:
   * The package extracts Purchase Order (PO) and Sales Order (SO) details.
   * It updates a staging table with relevant information.
   * Parameters include p\_frequency (DAILY or SPECIAL).
3. **Cursor Definitions**:
   * Two cursors (newbuy\_ext and allocation\_ext) retrieve relevant data from the staging table.
   * Filtering criteria include process flags and specific recommendation types.
4. **Processing Logic**:
   * For each record in newbuy\_ext:
     + Validates PO line location.
     + Updates staging table with relevant details.
     + Handles errors related to ASL (Approved Supplier List) and item revisions.
   * For each record in allocation\_ext:
     + Validates SO line ID.
     + Updates staging table with relevant details.
     + Handles errors related to requisitions and ASL.
5. **Purging Records**:
   * Another procedure (PURGING\_RECOMMENDATIONS) purges records older than 7 days from the staging table.
6. **Error Handling**:
   * Proper error messages are set for failed processes.
   * The package maintains process status and logs.

Remember, this is a high-level summary. If you need further details or have specific questions, feel free to ask!

Certainly! Let’s delve into the PL/SQL code to identify the queries responsible for extracting Purchase Order (PO) and Sales Order (SO) details. Here’s what I found:

1. **Extracting PO Details**:
   * The relevant cursor is named newbuy\_ext.
   * The query within this cursor retrieves the following information:
     + req\_line\_id: Requisition line ID.
     + transaction\_id: Transaction ID.
     + plan\_recommend: Plan recommendation type.
     + load\_id: Load ID.
     + item\_number: Item number.
     + destination\_organization\_code: Destination organization code.
2. **Extracting SO Details**:
   * The cursor responsible for SO details is named allocation\_ext.
   * The query within this cursor retrieves the following information:
     + req\_line\_id: Requisition line ID.
     + transaction\_id: Transaction ID.
     + plan\_recommend: Plan recommendation type.
     + load\_id: Load ID.
3. **Additional Notes**:
   * Both cursors filter records based on the process\_flag being equal to ‘P’.
   * The extracted details are then used to update a staging table.

Remember, this is a high-level summary. If you need further details or have specific questions about any part of the code, feel free to ask! 😊

gems\_po.gehc\_svc\_pln\_spm\_stg