ETL_process_AandB.txt
File

In the ETL_process_AandB.txt file there is a T-SQL stored procedure named ETL_proc_by_hand_source_A that loads data from ETL_source_A into ETL_destination. Please create a similar procedure named ETL_proc_Gen_source_B that loads data from ETL_source_B into ETL_destination. The column names are not identical, but are similar.

Thought for 1m 56s

Expand for details

To address your query, I will create a T-SQL stored procedure named `ETL_proc_Gen_source_B` that loads data from `ETL_source_B` into `ETL_destination`, mirroring the functionality of the existing procedure `ETL_proc_by_hand_source_A` which handles `ETL_source_A`. Since the column names between `ETL_source_A` and `ETL_source_B` are similar but not identical, I will map them appropriately while maintaining the core logic of the original procedure.

Step 1: Understand the Existing Procedure

The stored procedure `ETL_proc_by_hand_source_A` performs the following tasks:

- **Input:** Accepts a parameter `@batch_id INT` to specify which batch of data to process.
- **Temporary Table:** Creates a temp table `#attribute_names` to store distinct `attribute_name` values from `ETL_source_A` for the given `batch_id`, along with their corresponding `attribute_id` retrieved using the function `ETL_lookup_attribute_id`.
- **Loop Logic:** Iterates over each distinct `attribute_name`, and for each iteration, inserts rows into `ETL_destination` where the `value` can be converted to a `float`, using the `attribute_id`, `timestamp` (converted to `DATETIME`), and `value` (converted to `float`).
- **Destination Columns:** Inserts into `ETL_destination` with columns `attribute_id`, `event_dt`, and `event_value`.

However, the original procedure has a potential oversight: the `INSERT` statement does not filter by `attribute_name`, meaning it inserts all rows for the `batch_id` with the `attribute_id` of the current loop iteration, which could lead to incorrect data mapping. For accuracy, I'll assume this is unintended and correct it in the new procedure, but I'll align with the "similar procedure" requirement by maintaining its structure.

Step 2: Analyze Table Structures and Map Columns

Let's compare the source tables and map the columns to the destination:

- **ETL_source_A:**
  - `[batch_id] [int] NOT NULL`
  - `[attribute_name] [nvarchar](50) NOT NULL`
  - `[timestamp] [datetime] NOT NULL`
  - `[value] [nvarchar](30) NULL`
- **ETL_source_B:**
  - `[batch_id] [int] NOT NULL`
  - `[attribute] [nvarchar](50) NOT NULL`
  - `[when] [datetime] NOT NULL`
  - `[input_value] [nvarchar](30) NULL`
- **ETL_destination:**
  - `[attribute_id] [int] NOT NULL`
  - `[event_dt] [datetime] NOT NULL`
  - `[event_value] [float] NOT NULL`

Column Mapping for ETL_source_B:

- `[attribute]` corresponds to `[attribute_name]` in `ETL_source_A` and is used to lookup `attribute_id` via `ETL_lookup_attribute_id`.
- `[when]` maps to `event_dt` (both are `datetime`, so no conversion is needed, unlike the redundant `CONVERT` in the original).
- `[input_value]` maps to `event_value`, requiring conversion to `float` with validation.

Step 3: Design the New Procedure

The new procedure, `ETL_proc_Gen_source_B`, will:

- Take `@batch_id INT` as input.
- Use a temporary table to store distinct `[attribute]` values from `ETL_source_B` and their `attribute_id` values.
- Loop through each attribute, inserting corresponding rows into `ETL_destination`.
- Filter by both `batch_id` and `[attribute]` in the `INSERT` statement to ensure data accuracy.
- Use `TRY_CONVERT(float, [input_value])` to insert only valid numeric values, consistent with the original.

Solution: T-SQL Code for ETL_proc_Gen_source_B

```sql
CREATE OR ALTER PROC [dbo].[ETL_proc_Gen_source_B] @batch_id INT
AS
SET NOCOUNT ON

-- Declare variables
DECLARE @max_cnt INT,
        @cnt INT,
        @attribute NVARCHAR(50),
        @attribute_id INT

-- Create temporary table for distinct attributes
CREATE TABLE #attributes (
    cnt INT NOT NULL IDENTITY(1,1),
    attribute NVARCHAR(50) NOT NULL PRIMARY KEY CLUSTERED,
    attribute_id INT NOT NULL
)

-- Populate temp table with distinct attributes and their IDs
INSERT INTO #attributes (attribute, attribute_id)
SELECT DISTINCT [attribute], dbo.ETL_lookup_attribute_id([attribute])
FROM dbo.ETL_source_B
WHERE batch_id = @batch_id

-- Get the maximum count for the loop
SELECT @max_cnt = MAX(cnt) FROM #attributes
SET @cnt = 0

-- Loop through each attribute
WHILE @cnt < @max_cnt
BEGIN
    SET @cnt = @cnt + 1
```

```
    -- Get the current attribute and its ID
    SELECT @attribute = attribute,
           @attribute_id = attribute_id
    FROM #attributes
    WHERE cnt = @cnt

    -- Insert rows into ETL_destination, filtering by attribute
    INSERT INTO dbo.ETL_destination (attribute_id, event_dt, event_value)
    SELECT @attribute_id,
           [when],
           TRY_CONVERT(FLOAT, [input_value])
    FROM dbo.ETL_source_B
    WHERE TRY_CONVERT(FLOAT, [input_value]) IS NOT NULL
      AND batch_id = @batch_id
      AND [attribute] = @attribute
END
```

Explanation of the Code

- **Settings:** `SET NOCOUNT ON` prevents row count messages, matching the original.
- **Temp Table:** `#attributes` stores distinct `[attribute]` values and their `attribute_id` values, using `IDENTITY` for loop indexing.
- **Loop:** Iterates over each distinct `[attribute]`, retrieving the current `@attribute` and `@attribute_id`.
- **Insert Logic:**
  - `attribute_id`: Uses `@attribute_id` from the lookup function.
  - `event_dt`: Uses `[when]` directly (no conversion needed as it's already `datetime`).
  - `event_value`: Converts `[input_value]` to `float` with `TRY_CONVERT`, filtering out non-numeric values.
  - **Filter:** Includes `AND [attribute] = @attribute` to ensure only rows matching the current attribute are inserted, correcting the potential oversight in the original.

Notes

- **Assumption:** The `ETL_lookup_attribute_id` function works with `[attribute]` from `ETL_source_B` as it does with `[attribute_name]` from `ETL_source_A`,

assuming the `attribute` table's `attribute_name` column matches these values.

- **Improvement:** While a set-based approach (e.g., joining `ETL_source_B` with `attribute`) could be more efficient, I've retained the loop-based structure to align with `ETL_proc_by_hand_source_A`.
- **Robustness:** If an `[attribute]` doesn't exist in the `attribute` table, the function may return `NULL`, potentially causing issues. The original doesn't handle this, so I haven't added extra checks, but in a production environment, you might want to validate this.

This procedure successfully loads data from `ETL_source_B` into `ETL_destination`, respecting the column differences and maintaining the intended ETL process.