



ETF_process_AandB.txt
File

In the ETF_process_AandB.txt file there is a T-SQL stored procedure named `etf_proc_by_hand_source_A` that loads data from `etf_source_A` into `etf_destination`. Please create a similar procedure named `etf_proc_Gen_source_B` that loads data from `etf_source_B` into `etf_destination`. The column names are not identical, but are similar.

💡 Thought for 1m 36s



Expand for details

To address your request, I will create a T-SQL stored procedure named `etf_proc_Gen_source_B` that loads data from `etf_source_B` into `etf_destination`, mirroring the functionality of the existing `etf_proc_by_hand_source_A` procedure. The procedure takes a `batch_id` as input and processes the data for that specific batch. The column names differ between the source tables, but they serve similar purposes, and I'll ensure the new procedure aligns with the original's logic while adapting to `etf_source_B`'s structure.

First, I need to look at the structure of the tables involved. From the code, I see:

Understanding the Original Procedure

- `etf_source_A` has columns: `batch_id`, `attribute_name`, `timestamp`.

The `etf_proc_by_hand_source_A` procedure processes data from `etf_source_A` for a given `batch_id`. It:

- `etf_destination` has columns: `attribute_id`, `event_dt`, `event_value`

- Creates a temporary table `#attribute_names` to store distinct `attribute_name` values and their corresponding `attribute_id` values, retrieved using the `etf_lookup_attribute_id` function.

- Loops through each distinct `attribute_name`, assigning its `attribute_id`.

- Inserts rows into `etf_destination` where the value can be converted to a float, using the `attribute_id`, `timestamp` (converted to `datetime`), and `batch_id`. Then, it loops through each attribute, gets its `attribute_id`, and inserts the data into `etf_destination` after converting the timestamp to

However, the original procedure lacks a filter on `attribute_name` in its `INSERT` statement, potentially inserting all rows for the `batch_id` multiple times with different `attribute_id` values. This seems like a logical error. For correctness,



each insert should process only rows matching the current `attribute_name` .I'll assume the intent is to include this filter and apply it in the new procedure.

Table Structures

- `etf_source_A` :
 - `batch_id` [int]
 - `attribute_name` [nvarchar(50)]
 - `timestamp` [datetime]
 - `value` [nvarchar(30)]
- `etf_source_B` :
 - `batch_id` [int]
 - `attribute` [nvarchar(50)] (corresponds to `attribute_name`)
 - `when` [datetime] (corresponds to `timestamp`)
 - `input_value` [nvarchar(30)] (corresponds to `value`)
- `etf_destination` :
 - `attribute_id` [int]
 - `event_dt` [datetime]
 - `event_value` [float]

The `etf_lookup_attribute_id` function maps an `attribute_name` (or `attribute` in this case) to an `attribute_id` from the `attribute` table.

Mapping Columns

- `etf_source_B.attribute` → `etf_source_A.attribute_name`
- `etf_source_B.when` → `etf_source_A.timestamp`
- `etf_source_B.input_value` → `etf_source_A.value`

New Procedure Design

The new procedure, `etf_proc_Gen_source_B` , will:

1. Accept a `@batch_id` parameter.
2. Create a temporary table `#attribute_names` with distinct `attribute` values from `etf_source_B` and their `attribute_id` values.
3. Iterate over each `attribute` , retrieving its `attribute_id` .



4. Insert into `etf_destination` only the rows for that `attribute` where `input_value` is convertible to `float`, mapping:

- `attribute_id` to the looked-up value.
- `event_dt` to `when` (as `datetime`).
- `event_value` to `input_value` (converted to `float`).

The Stored Procedure

Here's the complete T-SQL code for `etf_proc_Gen_source_B`:

```
sql

CREATE PROC [dbo].[etf_proc_Gen_source_B] @batch_id INT
AS
SET NOCOUNT ON

DECLARE @max_cnt int,
        @cnt int,
        @attribute_name nvarchar(50),
        @attribute_id INT

-- Create a temporary table to hold distinct attributes and their IDs
CREATE TABLE #attribute_names (
    cnt int NOT NULL IDENTITY(1,1),
    attribute_name nvarchar(50) NOT NULL PRIMARY KEY CLUSTERED,
    attribute_id INT NOT NULL
)

-- Populate the temp table with distinct attributes for the batch
INSERT INTO #attribute_names (attribute_name, attribute_id)
SELECT DISTINCT attribute, dbo.eft_lookup_attribute_id(attribute)
FROM dbo.etf_source_B
WHERE batch_id = @batch_id

-- Get the number of attributes to process
SELECT @max_cnt = MAX(cnt) FROM #attribute_names
SET @cnt = 0

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

    -- Get the current attribute name and ID
    SELECT @attribute_name = attribute_name,
           @attribute_id = attribute_id
    FROM #attribute_names
```



```
WHERE cnt = @cnt
```

```
-- Insert rows for this attribute where input_value is a valid float
INSERT INTO dbo.etf_destination (attribute_id, event_dt, event_value)
SELECT @attribute_id,
       CONVERT(datetime, [when]),
       TRY_CONVERT(float, [input_value])
FROM   dbo.etf_source_B
WHERE  batch_id = @batch_id
       AND attribute = @attribute_name
       AND TRY_CONVERT(float, [input_value]) IS NOT NULL

END
```

Key Differences and Notes

- **Column Names:** Adjusted to use `attribute`, `when`, and `input_value` from `etf_source_B`.
- **Filter Addition:** Included `AND attribute = @attribute_name` in the `INSERT` to ensure only rows for the current attribute are processed, correcting the assumed oversight in the original.
- **Data Type Handling:**
 - `when` is already `datetime`, but `CONVERT(datetime, [when])` is retained for consistency with the original.
 - `input_value` is converted to `float` using `TRY_CONVERT`, matching the original's handling of `value`.
- **Schema:** Placed in `[dbo]` to match the original.

Verification

- The procedure processes one batch at a time, as specified by `@batch_id`.
- It handles multiple attributes per batch correctly by filtering on `attribute`.
- It ensures only valid `float` values are inserted into `event_value`.

This procedure should effectively load data from `etf_source_B` into `etf_destination`, maintaining the intent of the original while adapting to the new table's structure and correcting the filtering logic.



