

## **Dynamic Table Archiving – Overview**

This approach provides a **dynamic, parameterized solution** for archiving data from any table in a source database to a flat file target. Designed to work across various tables without requiring manual mapping for each table, it ensures a flexible and reusable process that adapts to different table structures with minimal configuration.

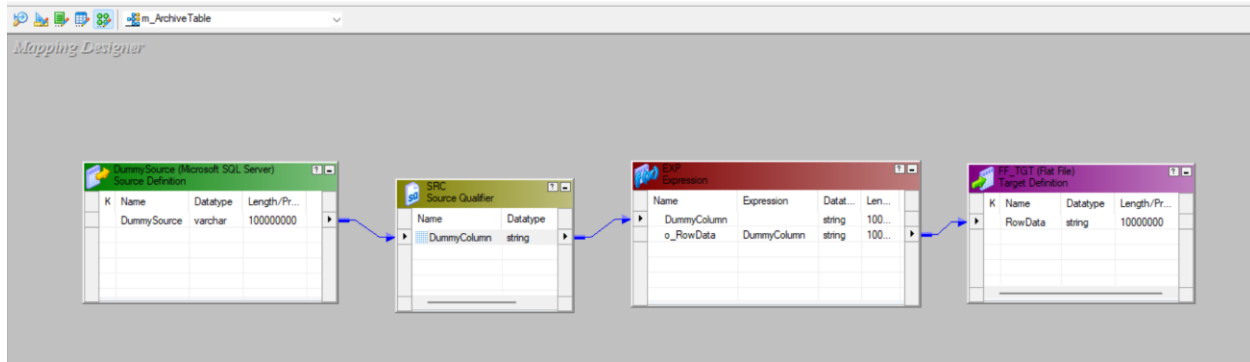
The primary objective is to enable **automated archiving** of tables with varying row sizes, column counts, and data types by passing key parameters at runtime, such as the table name, source connection, and target file path. This eliminates the need for individual workflows or mappings for each table, significantly reducing manual effort and maintenance overhead.

By leveraging Informatica PowerCenter's capabilities, the process dynamically handles data extraction, formatting, and output to a flat file while ensuring data integrity, handling large row sizes, and preserving null values. Core settings such as buffer management, line sequential handling, and memory allocation have been optimized to accommodate high-volume tables with wide rows, ensuring consistent performance across different data loads.

The workflow takes advantage of **parameter files**, which allow table-specific values to be supplied at runtime. This ensures that the same mapping and session can be reused for different tables without modifications. Output files are generated in a consistent format with proper delimiters, making them ready for downstream consumption or long-term storage.

This dynamic approach is particularly suited for environments where frequent table archiving is required, and the table schema can vary over time. It streamlines the overall process, ensures scalability, and provides a foundation for automated data archiving solutions.

# Overview of Technical Steps – Mapping Creation (Designer Phase)



Declare Parameters and Variables

Declare the parameters and variables you want to use in the mapping or maplet.  
A parameter represents a constant value defined before mapping run.  
A variable represents a value that can be changed during mapping run.

Name	Type	Datatype	Prec	Scale	Aggregation	IsExprVar
\$\$TableName	Param...	string	1...	0		FALSE

Initial value:

Description:

OK Cancel Help

## 1. Source Definition

- A **dummy source definition** was created using Microsoft SQL Server, with a single column named DummySource of data type **varchar** and precision **10,000,000** (10 MB) to handle large concatenated rows dynamically.
- This ensures that the mapping can process any table structure without schema dependency.

*(Refer to Screenshot 1)*

## 2. Source Qualifier

- A **Source Qualifier (SQ)** transformation was added and linked to the dummy source.

- The column DummyColumn was configured as a **string** with appropriate precision to pass all data rows for further transformation.

*(Refer to Screenshot 1)*

### 3. Expression Transformation

- An **Expression (EXP)** transformation was used to prepare the final output row by concatenating columns into a single string output.
- An output port named o\_RowData was created with a precision of **10,000,000** to ensure large rows are handled without truncation.
- The expression for o\_RowData was set to pass the value of DummyColumn directly.

*(Refer to Screenshot 1)*

### 4. Flat File Target Definition

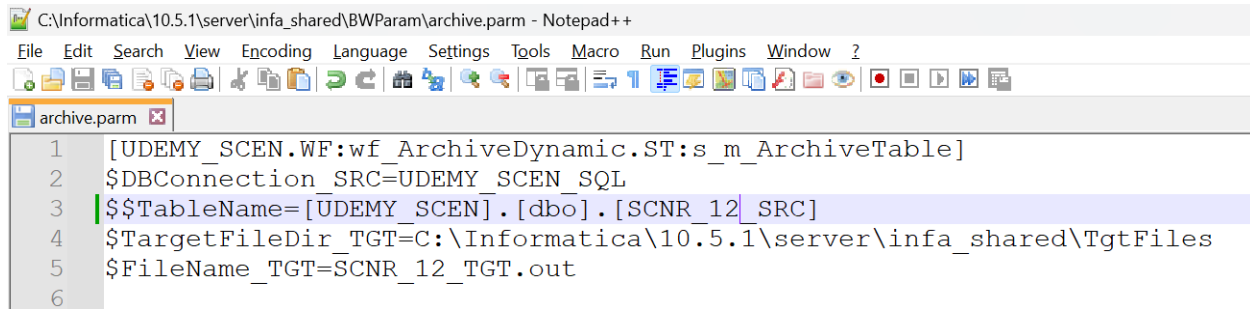
- A **Flat File Target (FF\_TGT)** was defined with a single column named RowData of type **string** and precision **10,000,000**.
- This target is designed to capture the entire row data as a single line in the output flat file.

*(Refer to Screenshot 1)*

### 5. Parameter Declaration

- A mapping parameter named **\$\$TableName** was declared to dynamically pass the source table name at runtime.
- The parameter type is **string** with a precision of **100** to accommodate long table names if necessary.

# Overview of Parameter File – archive.param



```
1 [UDEMY_SCEN.WF:wf_ArchiveDynamic.ST:s_m_ArchiveTable]
2 $DBConnection_SRC=UDEMY_SCEN_SQL
3 $$TableName=[UDEMY_SCEN].[dbo].[SCNR_12_SRC]
4 $TargetFileDir_TGT=C:\Informatica\10.5.1\server\infa_shared\TgtFiles
5 $FileName_TGT=SCNR_12_TGT.out
6
```

The parameter file archive.param is essential in enabling a fully **parameterized and dynamic workflow**. It allows runtime values to be passed for key parameters, ensuring that the workflow and mapping can handle different tables and targets without requiring manual modifications.

## 1. Header Section

**[UDEMY\_SCEN.WF:wf\_ArchiveDynamic.ST:s\_m\_ArchiveTable]**

This header specifies the **workflow (wf\_ArchiveDynamic)** and the **session (s\_m\_ArchiveTable)** that will use the defined parameters. It ensures PowerCenter applies these values only when this specific workflow and session are executed.

## 2. Source Database Connection Parameter

**\$DBConnection\_SRC=UDEMY\_SCEN\_SQL**

Defines the **source database connection** to be used during execution. Parameterizing the connection allows reusability across multiple environments (e.g., development, testing, production) by simply updating this value.

## 3. Source Table Name Parameter

**\$\$TableName=[UDEMY\_SCEN].[dbo].[SCNR\_12\_SRC]**

Specifies the **source table name** to be archived. This parameter enables dynamic table selection, making it possible to archive different tables without altering the mapping or session configuration.

## 4. Target File Directory Parameter

**\$TargetFileDir\_TGT=C:\Informatica\10.5.1\server\infa\_shared\TgtFiles**

Defines the **target directory path** where the archived data will be stored. Parameterizing the directory path allows flexible control over output locations.

## **5. Target File Name Parameter**

**\$FileName\_TGT=SCNR\_12\_TGT.out**

Specifies the **name of the target flat file**. This ensures that the output file can be uniquely named for each table being archived.

## **How It Supports Complete Parameterization**

### **1. Dynamic Table Selection:**

The parameter \$\$TableName allows the same mapping to archive any table by passing its name at runtime.

### **2. Flexible Target Output:**

Parameterizing the target directory and file name ensures that outputs can be redirected to different locations with appropriate file names.

### **3. Reusable Workflow:**

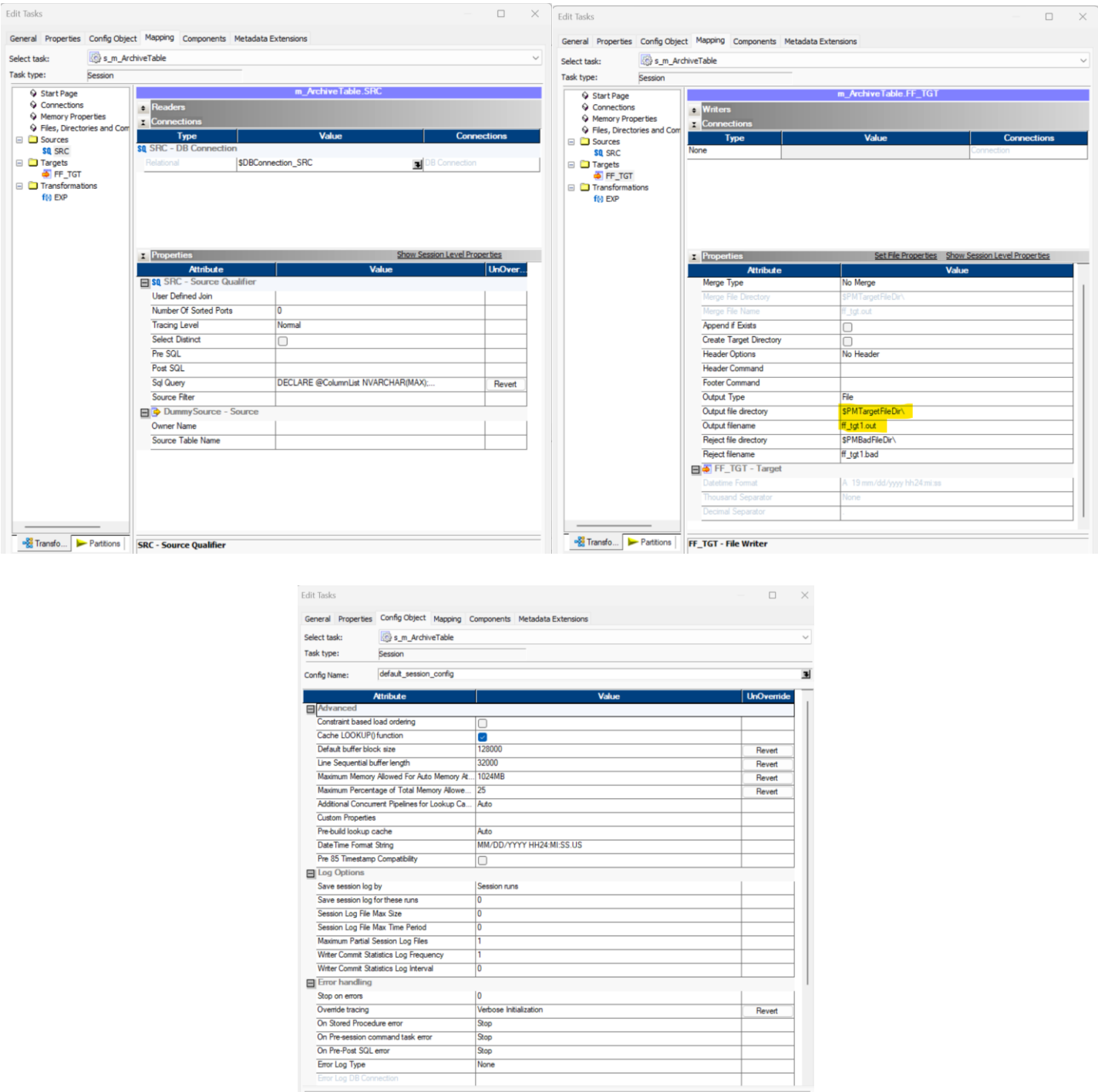
By parameterizing the database connection, the workflow can be reused across multiple environments by simply modifying the connection parameter in the file.

### **4. Reduced Maintenance Effort:**

Since runtime values are supplied via the parameter file, there is no need to modify the workflow or mapping when switching between different tables or environments, significantly reducing maintenance overhead.

This parameterized approach ensures a **scalable**, **flexible**, and **low-maintenance** solution for dynamically archiving tables in PowerCenter.

# Workflow and Session Properties Overview



The session **s\_m\_ArchiveTable** within the workflow **wf\_ArchiveDynamic** has been designed to dynamically handle different tables and generate flat file outputs. The session properties are configured to use runtime parameters for both the **source database connection** and the **target file output**, ensuring a high level of flexibility and reusability.

## SQL Query for Dynamic Column Concatenation (Detailed with Comments)

```
DECLARE @ColumnList NVARCHAR(MAX); -- Declare a variable to hold the
concatated column list

DECLARE @SQL NVARCHAR(MAX);          -- Declare a variable to hold the dynamic SQL
query

-- Generate a list of columns, replacing nulls and casting each column to
NVARCHAR(MAX)

SELECT @ColumnList = STRING_AGG('ISNULL(CAST(' + QUOTENAME(name) + ' AS
NVARCHAR(MAX)), ''')', ' + ', '' + ')

FROM sys.columns

WHERE object_id = OBJECT_ID('$$TableName'); -- Dynamically fetch columns for the
table passed via parameter

-- Build the dynamic SQL to concatenate all columns for each row

SET @SQL = 'SELECT ' + @ColumnList + ' AS RowData FROM $$TableName'; --
Construct the final SQL query

-- Execute the dynamic SQL

EXEC sp_executesql @SQL; -- Execute the constructed query
```

### Explanation:

1. **Column Handling:** The query dynamically retrieves all columns from the specified table (provided as a parameter \$\$TableName) and converts them to strings.
2. **Null Handling:** ISNULL ensures that null values are replaced with empty strings to avoid null output in the concatenated result.
3. **Dynamic Execution:** The query uses sp\_executesql to execute the dynamically constructed SQL, ensuring that the approach works for any table structure passed at runtime.

### Parametrization of Source and Target

#### 1. Source Database Connection:

The **source connection** is fully parameterized using the parameter \$DBConnection\_SRC, which points to the database connection defined in the parameter file. This ensures that the session can dynamically connect to different databases without modifying the workflow.

## 2. Source Table Name:

The table name is passed dynamically via the parameter \$\$TableName. This allows the session to archive any table specified at runtime without altering the mapping or session.

## 3. Target File and Directory:

Although the intended goal was to fully parameterize the target file path using \$TargetFileDir\_TGT and \$FileName\_TGT, the correct syntax for using these parameters dynamically is still **work in progress**.

**(Highlighted area in screenshot)**

## Buffer Settings and Their Benefits

- **Default Buffer Block Size:** Set to **128KB**, this defines the size of each buffer block for reading and writing data.
- **Line Sequential Buffer Length:** Set to **32,000**, ensuring that sufficient memory is allocated for handling large rows when reading from the source or writing to the target.
- **Maximum Memory Allowed for Auto Memory Allocation:** Set to **1024MB**, this ensures efficient memory usage, reducing the frequency of disk I/O operations and improving session performance.

Increasing these buffer settings improves the session's ability to handle large datasets efficiently, minimizes I/O bottlenecks, and ensures smoother data flow during execution.

## Limitations of this Approach:

### Maximum String Size Limitation

- **PowerCenter limitation:** The string data type in PowerCenter allows a maximum precision of **10,000,000 characters per row**.
- **Implication:**
  - If the concatenated data for a single row exceeds this limit, the data will be truncated, resulting in **data loss**.



- This limitation makes the approach less suitable for archiving very wide tables with a large number of columns or columns containing large text data.
- **Workaround:** Currently, there is no direct way to handle rows exceeding this limit within PowerCenter. A potential enhancement could involve splitting large rows into multiple smaller parts or using a compressed format before writing to the flat file.

## **Future Improvements**

To overcome PowerCenter's string size limitation of 10,000,000 characters per row, a potential enhancement involves splitting large rows into multiple smaller ones, thereby allowing more efficient file archiving. This can be achieved by implementing a SQL-based preprocessing step that dynamically partitions the incoming database records into smaller segments before they are processed by Informatica PowerCenter.

### **Proposed SQL Enhancement for Row Splitting**

A refined SQL query could break down large concatenated rows into multiple rows, ensuring that each output file contains a manageable portion of data. This would allow X number of rows to be evenly distributed into X separate files, effectively bypassing the string limit constraint.

- **Dynamic Row Partitioning:** The SQL query will generate a sequentially numbered partition for each data chunk, ensuring that wide rows are split logically while maintaining referential integrity.
- **File Distribution Mechanism:** By associating partitioned data with unique file identifiers, Informatica can generate multiple output files for the same input dataset, facilitating better storage management and downstream processing.
- **Optimized Performance:** This approach reduces memory overhead and minimizes the risk of truncation errors, enhancing overall system stability and efficiency.

This enhancement would significantly improve the robustness of the archiving process, ensuring compatibility with datasets that exceed PowerCenter's row size limits while maintaining a scalable and automated data processing pipeline.