**SSIS Row Count Transformation – Trainer Notes (End‑to‑End with Foreach, Flat File, OLE DB, and Auditing)**

# 1) What the Row Count Transformation Does

Row Count is a Data Flow transformation that counts the number of rows that pass through it and stores the value in an SSIS variable. It is commonly used for auditing (per‑file row counts, totals, email notifications) and for conditional control flow.

# 2) When to Use It

* When loading from non‑SQL sources (CSV, Excel, JSON via script) where you can’t easily run COUNT(\*) beforehand.
* When you need per‑file counts in Foreach File loops.
* When you want to log counts into an audit table, a text file, or send them via email.

# 3) Demo Scenario Overview

We will load three CSV files (TestData\_1.csv, TestData\_2.csv, TestData\_3.csv) into a SQL Server table [dbo].[TestData]. We’ll capture the number of data rows for each file and insert FilePath + RowCount + Date into an audit table [dbo].[Logs]. The Flat File Source is configured to ignore the header row, so the Row Count value equals the number of data rows inserted.

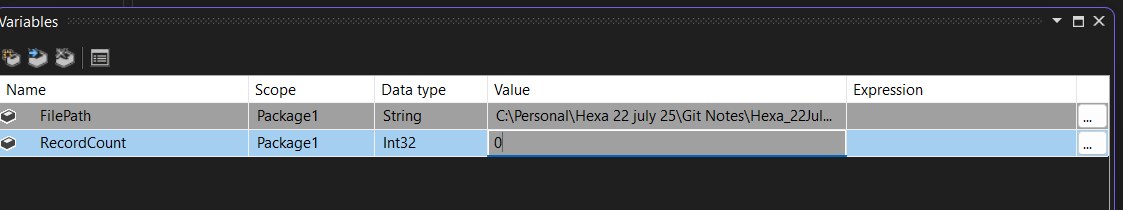
# 4) Prereqs – Tables and Sample Script

## Create destination and audit tables

```sql  
CREATE TABLE dbo.TestData(  
 ID varchar(50) NULL,  
 first\_name varchar(50) NULL,  
 last\_name varchar(50) NULL,  
 gender varchar(50) NULL,  
 company\_name varchar(50) NULL  
);  
  
CREATE TABLE dbo.Logs(  
 ID int IDENTITY(1,1) PRIMARY KEY,  
 FilePath varchar(300) NOT NULL,  
 RecordCount int NOT NULL,  
 Dated datetime NOT NULL DEFAULT(getdate())  
);  
```

# 5) SSIS Variables (Package Scope)

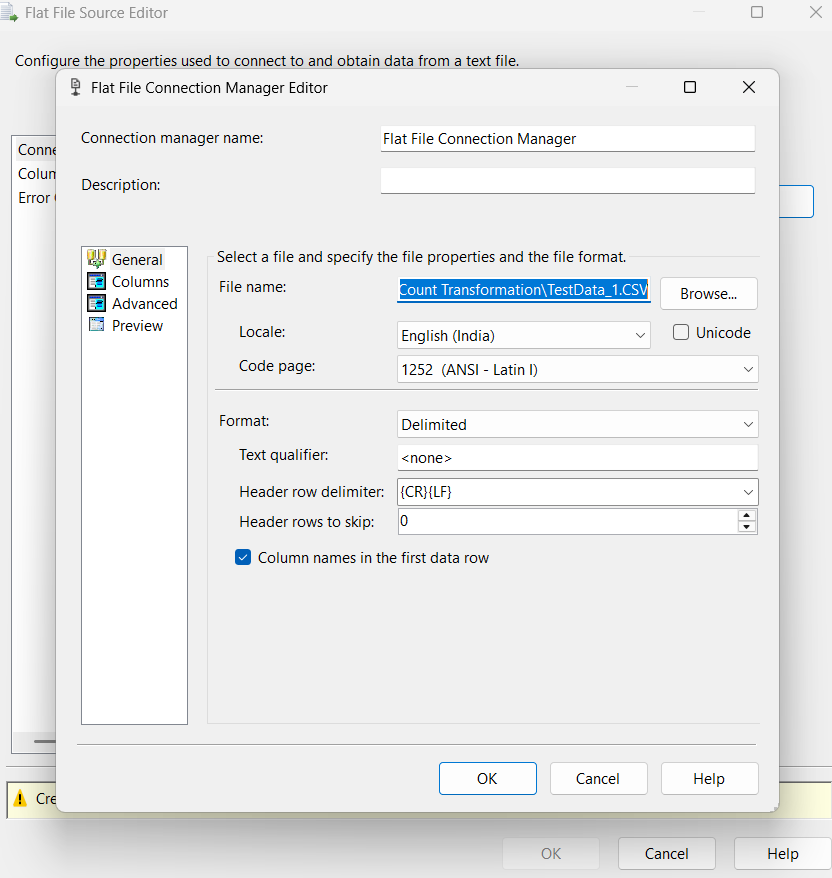
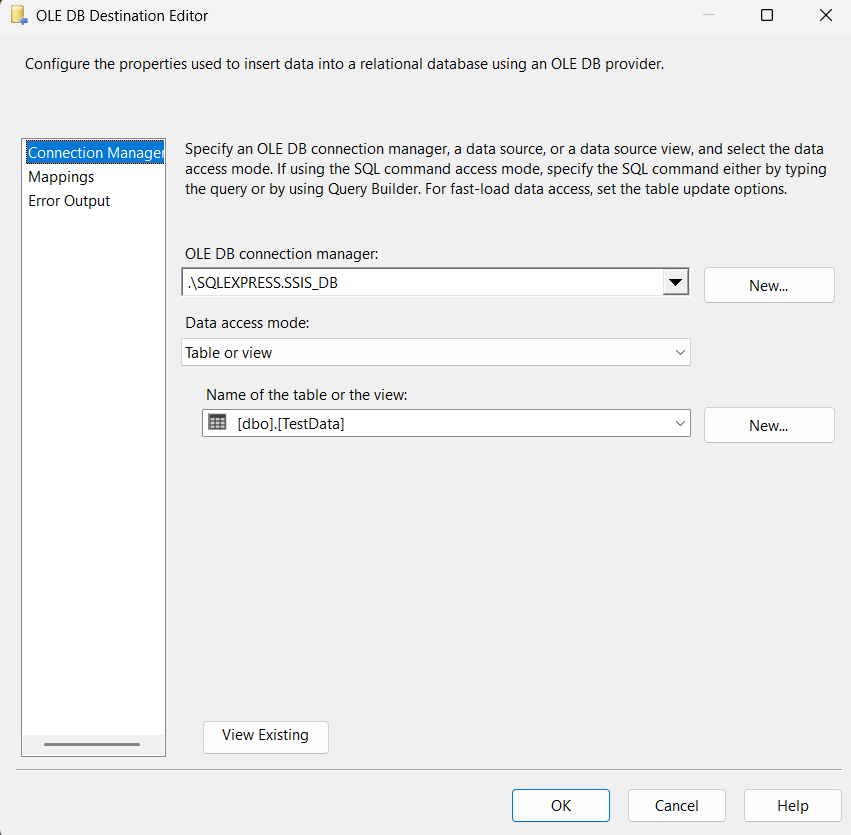
* User::FilePath – String – full path of the current CSV file (e.g., C:\\Files\\TestData\_1.csv).
* User::RecordCount – Int32 – will be set by Row Count transformation.



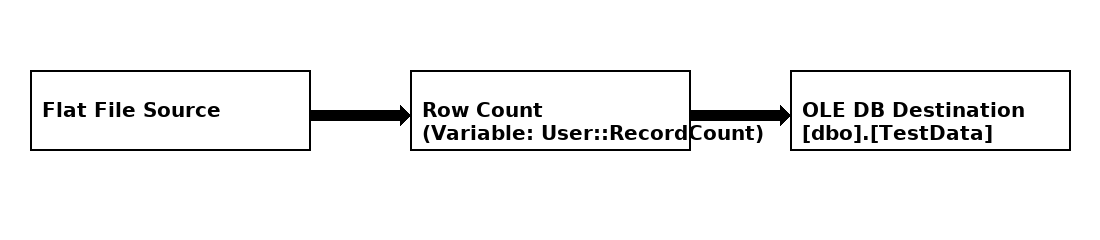
# 6) Control Flow – Foreach Loop to Iterate Files

1. Drag \*\*Foreach Loop Container\*\*.
2. Collection → \*\*Foreach File Enumerator\*\* → Folder: `C:\Files`, Files: `\*.csv`, Retrieve file name: \*\*Fully qualified\*\*.
3. Variable Mappings → Index 0 → \*\*User::FilePath\*\*.
4. Inside the loop, add a \*\*Data Flow Task\*\* (Load CSV).

# 7) Data Flow – Source → Row Count → Destination

1. Add \*\*Flat File Source\*\* and create a Flat File Connection Manager using one of the CSV files for metadata.
2. In the connection manager, tick \*\*Column names in the first data row\*\* (this excludes header from the pipeline).
3. 
4. Make the connection dynamic: Connection Manager → \*\*Properties → Expressions → ConnectionString = @[User::FilePath]\*\*. Also set the \*\*Data Flow Task → DelayValidation = True\*\*.
5. Add \*\*Row Count\*\* transformation between source and destination. Double‑click Row Count and set \*\*VariableName = User::RecordCount\*\*.
6. Add \*\*OLE DB Destination\*\* → connect to your database and map columns to \*\*dbo.TestData\*\*.
7. 

## Pipeline Diagram



# 8) Log the Count per File – Execute SQL Task (Best Practice: Parameters)

After the Data Flow, add an \*\*Execute SQL Task\*\* named “Log file info”. Use a parameterized INSERT to avoid string concatenation issues.

## Execute SQL Task Settings

* ConnectionType: OLE DB
* SQLStatement: INSERT INTO dbo.Logs(FilePath, RecordCount, Dated) VALUES (?, ?, GETDATE());
* ResultSet: None
* BypassPrepare: False (default)
* Parameter Mapping:

• Parameter 0 → Variable: User::FilePath, Direction: Input, Data type: VARCHAR, Size: 300  
• Parameter 1 → Variable: User::RecordCount, Direction: Input, Data type: LONG (Int32)

## Alternate (Expression-Built SQL – not recommended)

If you must use an expression, set Execute SQL Task → Expressions → SqlStatementSource to:  
```  
" INSERT INTO Logs values ( '"+ @[User::FilePath] +"',"+(DT\_WSTR,12) @[User::RecordCount] +",GETDATE()) "  
```  
Parameters are preferred for safety and simplicity.

# 9) Fix Checklist for Your Package (27 RowCount TransformationDemo.dtsx)

* Flat File \*\*Connection Manager\*\* has \*\*Expressions → ConnectionString = @[User::FilePath]\*\*.
* Data Flow Task \*\*DelayValidation = True\*\*; Flat File Source \*\*ValidateExternalMetadata = False\*\*.
* Row Count sits between Source and Destination and writes to \*\*User::RecordCount (Int32, package scope)\*\*.
* Execute SQL Task uses \*\*parameters\*\* (not an expression string) and is connected with \*\*Success\*\* from the Data Flow.
* Foreach enumerator returns \*\*Fully qualified\*\* file names and maps \*\*Index 0 → User::FilePath\*\*.
* There is only \*\*one\*\* Flat File Connection Manager referenced by the source (delete duplicates).

# 10) Common Pitfalls

* Expression set on the \*\*source\*\* instead of the \*\*connection manager\*\* (source keeps reading the same file).
* Wrong variable bound (e.g., `User::RecordCount`) leading to errors like “Cannot open datafile '0'”.
* Reading `User::RecordCount` too early; fix with parameterized SQL and Success precedence.

# 1) Flat File Connection Manager → Expression: ConnectionString = @[User::FilePath]

**What it is:** You’re binding the file path of the connection **manager** to a variable.  
**Why:** In a Foreach loop you want to read a different file each iteration. Setting this expression means the connection points to whatever User::FilePath is at runtime (e.g., C:\Data\TestData\_1.csv, then ...\_2.csv, etc.).  
**Without it:** The source keeps reading the same file you picked at design time.  
**Tip:** Put the expression on the **connection manager**, not on the Flat File **Source**. That’s the classic reason for “it logs new names but always loads the first file.”

# 2) Flat File Source → ValidateExternalMetadata = False

**What it is:** Tells the source **not** to verify the external file/columns at design-time or pre-validate.  
**Why:** With a dynamic ConnectionString, the specific file may not exist yet (or may change) when SSIS tries to validate. Setting this to **False** avoids early failures like “cannot open datafile…”.  
**Trade-off:** You give up a bit of early safety. If the file is truly missing or malformed, you’ll discover it **at runtime** (which is fine in a Foreach scenario).  
**When to turn back ON:** Fixed paths, stable schemas—validation helps catch mistakes earlier.

# 3) Data Flow Task → DelayValidation = True

**What it is:** Defers the task’s validation until it actually runs.  
**Why:** Similar reason—when connections/queries are built from variables or expressions, early validation can fail because the objects aren’t ready yet. Delaying validation lets your Foreach set User::FilePath first, then the Data Flow validates against the **current** file.  
**Without it:** The package can fail before the loop assigns the right path, or it sticks to the first evaluated file.

# 4) Row Count → writes to User::RecordCount (Int32)

**What it is:** A transform that **counts rows flowing through it** and stores that count into the variable you choose.  
**When it updates:** During the Data Flow execution; the variable holds the **final** count once the flow finishes.  
**Why:** Perfect for per-file auditing—after the Data Flow completes, you can read User::RecordCount (e.g., in an Execute SQL Task) and log the file name + row count.  
**Gotcha:** If you build the INSERT text via **expression** too early, you might read 0. Using **parameterized SQL** (or placing the log task after the Data Flow with a Success constraint) ensures you capture the updated value.

 **Expression on Connection Manager** → “Which file should I read **this** iteration?”

 **ValidateExternalMetadata = False** → “Don’t check the file until I actually run.”

 **DelayValidation = True** → “Don’t validate this Data Flow until the loop sets the right values.”

 **Row Count → variable** → “Count the rows that actually flowed and save that number for logging.”