**SSIS Aggregate Transformation – Trainer Notes (AdventureWorks 2014)**

# 1. What is the Aggregate Transformation?

The Aggregate transformation performs common aggregations (MIN, MAX, SUM, AVG, COUNT) and optional GROUP BY operations on rows flowing through a Data Flow. It is useful when your source is not a relational database (CSV, Excel, flat files, APIs) or when you want to keep all logic inside the SSIS pipeline.

# 2. When to Use vs. SQL Aggregates

* Use SQL (e.g., T‑SQL GROUP BY) when the source is a database and push‑down is possible for best performance.
* Use Aggregate transformation when data originates from non‑SQL sources (CSV/Excel) or when you must aggregate multiple sources after merges within the pipeline.
* Use it to compute several metrics in one pass (e.g., MIN Bonus, SUM CommissionPct, AVG SalesYTD, MAX SalesLastYear).

# 3. Sample Source and Fields (AdventureWorks 2014)

Table: Sales.SalesPerson  
Columns used in this demo:  
• Bonus (money) -min  
• CommissionPct (smallmoney) -sum  
• SalesYTD (money) -Avg  
• SalesLastYear (money) – max

You may think we can perform all the things in the sql query itself why we want to Go with aggregate transformation When we have a data from different sources like csv and other formats we can’t perform aggregate function using sql query in such case the aggregate transformation will be very helpful

# 4. Hands‑On: Build the Package in Visual Studio 2022

## A) Control Flow

1. Create a new Integration Services Project.
2. Drag a Data Flow Task and rename it to “Load Aggregates”. Double‑click to enter the Data Flow.

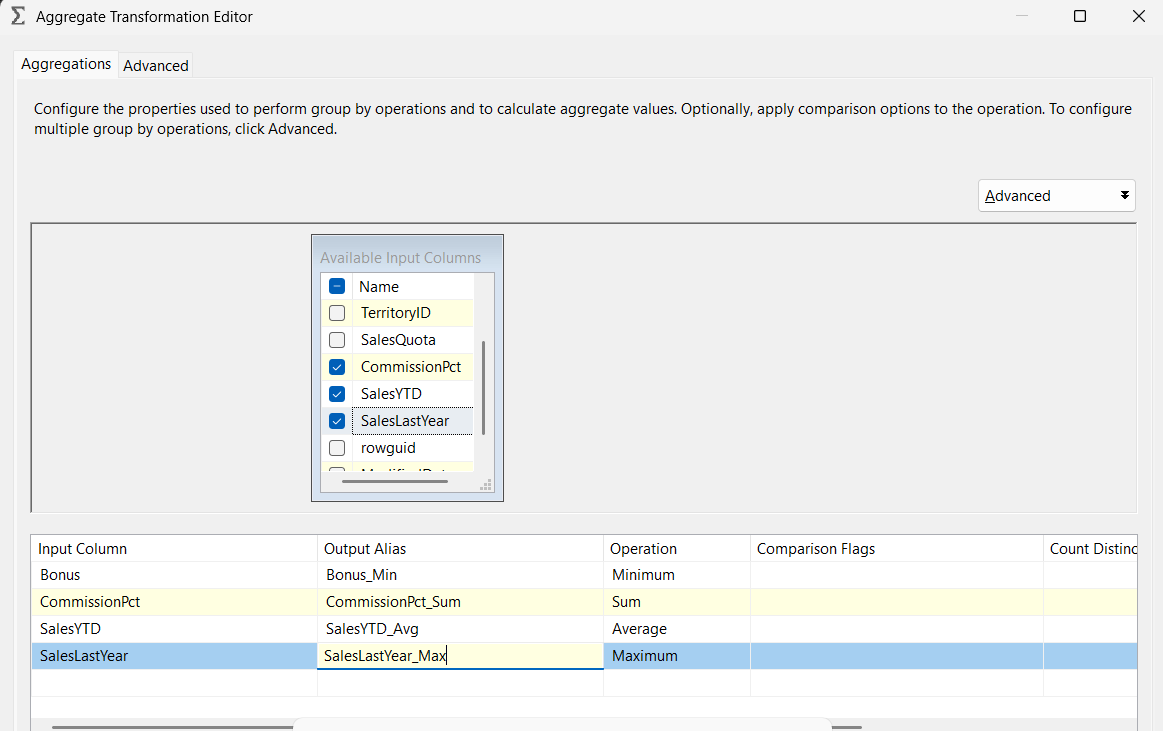
## B) Data Flow – Source

1. Add an OLE DB Source.
2. Create/choose an OLE DB Connection to your AdventureWorks2014 database.
3. In OLE DB Source, set Data access mode to “SQL command” and use:

SELECT \* FROM Sales.SalesPerson

## C) Data Flow – Aggregate Transformation

1. Drag the Aggregate transformation and connect the OLE DB Source (blue path).
2. Open Aggregate editor. In “Available Input Columns”, tick the columns you need to aggregate.
3. For each selected column, set Operation and Output Alias as follows:

* Bonus → Operation: Minimum → Output Alias: Bonus\_Min
* CommissionPct → Operation: Sum → Output Alias: Commission\_Sum
* SalesYTD → Operation: Average → Output Alias: SalesYTD\_Avg
* SalesLastYear → Operation: Maximum → Output Alias: SalesLastYear\_Max
* 

## E) Data Flow – Destination (Flat File)

1. Drag a Flat File Destination and connect it to Aggregate.
2. Create a new Flat File Connection Manager → Delimited.
3. Browse and set the file name (e.g., C:\Data\Aggregate.txt).
4. Tick “Column names in the first data row”.
5. Verify Mappings and click OK.

# 5. Run and Verify

1. Execute the package. Expect the OLE DB Source to read ~17 rows (depends on your data) and Aggregate to output a single row with the four metrics.
2. Open the output file and confirm values: Bonus\_Min, Commission\_Sum, SalesYTD\_Avg, SalesLastYear\_Max.

# 6. Real‑World Variations

* CSV/Excel inputs: Replace the OLE DB Source with Flat File/Excel Source; keep the Aggregate step unchanged.
* Multiple metrics in one pass: The Aggregate editor allows multiple output columns with different operations simultaneously.
* Per‑group rollups: Use Group by on one or more keys (e.g., TerritoryID, YearMonth) and compute SUM/AVG/MIN/MAX per group.
* Row counts: Either add an Aggregate with Operation=Count or place a Row Count transform to store counts in a variable.

# 7. Tips, Performance & Behavior

* The Aggregate transform is a blocking operation—it must see all rows for a group before producing output. Expect memory usage proportional to grouping cardinality.
* When the source is a database, push aggregations to SQL when possible for performance.
* NULL handling: For numeric operations like SUM/AVG/MIN/MAX, rows with NULLs in the aggregated column are ignored; COUNT counts rows that are not NULL for the chosen column.
* If your input is already sorted by the group keys (using Sort transform or an ORDER BY from source), set the IsSorted metadata and SortKeyPositions to reduce memory pressure.
* For dynamic file outputs, set Flat File Connection Manager → Expressions → ConnectionString = @[User::OutputPath] and set DelayValidation=True on the Data Flow Task.

# 8. Common Pitfalls & Fixes

* “Cannot open datafile '0'” → Your ConnectionString expression is bound to an Int32 (e.g., a Row Count variable). Bind it to your String path variable instead.
* Mismatched column metadata after changing the source → Reopen Aggregate and Destination, refresh metadata, and remap.
* Wrong totals due to duplicate rows → Add a Group by on natural keys first or deduplicate upstream before aggregating.

# 9. Validation Checklist

* Source preview shows expected numeric columns and no unexpected NULLs.
* Aggregate editor lists the correct operations and aliases.
* Destination mappings align with output columns.
* Package runs with expected input row count and expected 1 row (or N rows per group) at the destination.

# 10. Extension Exercises (Lab)

1. Add TerritoryID as Group by and compute SUM(SalesYTD) per territory. Write results to a table [dbo].[SalesTerritoryAgg].
2. Load the same four columns from a CSV instead of SQL and compute the same aggregates.
3. Add COUNT(\*) and COUNT of non‑NULL Bonus (by using Bonus as the input for a Count operation) to compare results.

# 11. Group By Variant: Territory-Level Metrics

This variant shows how to compute per-territory aggregates (one output row per TerritoryID). You will add a Group by key and then compute multiple measures in the same Aggregate transformation.

## Steps

1. Ensure the OLE DB Source also selects TerritoryID:  
   SELECT TerritoryID, Bonus, CommissionPct, SalesYTD, SalesLastYear FROM Sales.SalesPerson
2. Open Aggregate → tick TerritoryID and set Operation = Group by.
3. Add the following aggregate outputs:

* SalesYTD → SUM → Output Alias: SalesYTD\_Sum
* SalesLastYear → MAX → Output Alias: SalesLastYear\_Max
* CommissionPct → SUM → Output Alias: Commission\_Sum
* Bonus → MIN → Output Alias: Bonus\_Min
* Any column → COUNT → Output Alias: RowCount (optional)

1. Connect Aggregate to your destination (table or flat file).

## Create a destination table (if writing to SQL)

Use this table for the per-territory results:

CREATE TABLE dbo.SalesTerritoryAgg  
(  
 TerritoryID int NOT NULL,  
 SalesYTD\_Sum money NULL,  
 SalesLastYear\_Max money NULL,  
 Commission\_Sum smallmoney NULL,  
 Bonus\_Min money NULL,  
 RowCount int NULL  
);

## Mapping & Expectations

* Mappings: TerritoryID → TerritoryID; SalesYTD\_Sum → SalesYTD\_Sum; SalesLastYear\_Max → SalesLastYear\_Max; Commission\_Sum → Commission\_Sum; Bonus\_Min → Bonus\_Min; RowCount → RowCount.
* Expected rows: one per TerritoryID (cardinality of territories).
* Performance tip: If source can provide rows ordered by TerritoryID, set the source query with ORDER BY TerritoryID and mark the input as IsSorted to lower memory usage.

# 12. Screenshots (Illustrative)

Figure 1. Aggregate Transformation – No Group by (multiple measures in one row).

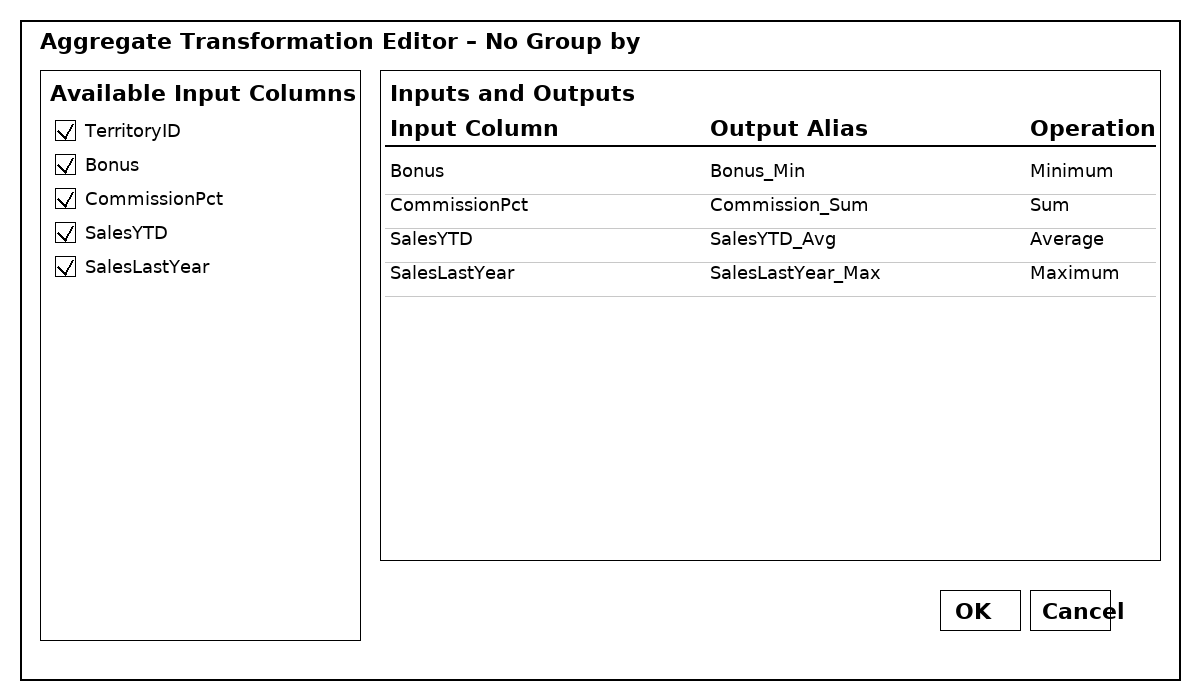


Figure 2. Aggregate Transformation – Group by TerritoryID (one row per TerritoryID).

