# **Understanding Data Movement in Azure Synapse SQL Pool**

Table of Contents

[**Understanding Data Movement in Azure Synapse SQL Pool** 1](#_Toc42001573)

[Introduction 2](#_Toc42001574)

[Why data moves 3](#_Toc42001575)

[Distribution Incompatible Join 3](#_Toc42001576)

[(\*) all queries are always refactored by the MPP DW Query Optimizer, even a single table SELECT statement. 3](#_Toc42001577)

[Distribution Incompatible Aggregation 4](#_Toc42001578)

[Query Syntax 4](#_Toc42001579)

[Re-distribute Data 4](#_Toc42001580)

[Round-Robin distributed tables 4](#_Toc42001581)

[Data Movement types 5](#_Toc42001582)

[Shuffle Move 5](#_Toc42001583)

[Broadcast Move 5](#_Toc42001584)

[Partition Move 5](#_Toc42001585)

[Move 5](#_Toc42001586)

[Trim Move 5](#_Toc42001587)

[Replicated tables 6](#_Toc42001588)

## Introduction

Azure Synapse SQL Pool is an MPP DW (Massively Parallel Processing) environment. It is assumed that the reader has an understanding of the basics architecture of the database, and of the table geometries available for use in the SQL Pool DW environment.

SQL Pool DW is implemented as a Scale-Out environment, where data and processing are handled by multiple resources. This is implemented in two dimensions – the data is sharded across 60 Distributions, and processing is sharded across 1 – 60 compute nodes.

This document is to describe the different data movement types that you may see in Azure Synapse SQL Pool DW, when you will see data movement, and the implications of data movement.

Note that DMS (Data Movement Service) is the legacy process moving data within the SQL Pool, this has been replaced by a direct SQL – SQL service, that eliminates some of the more costly steps in DMS. However, DMS is still used for certain moves and for external (Polybase) data movement. DMS may have been removed or its usage may have been modified, by the time you read this!

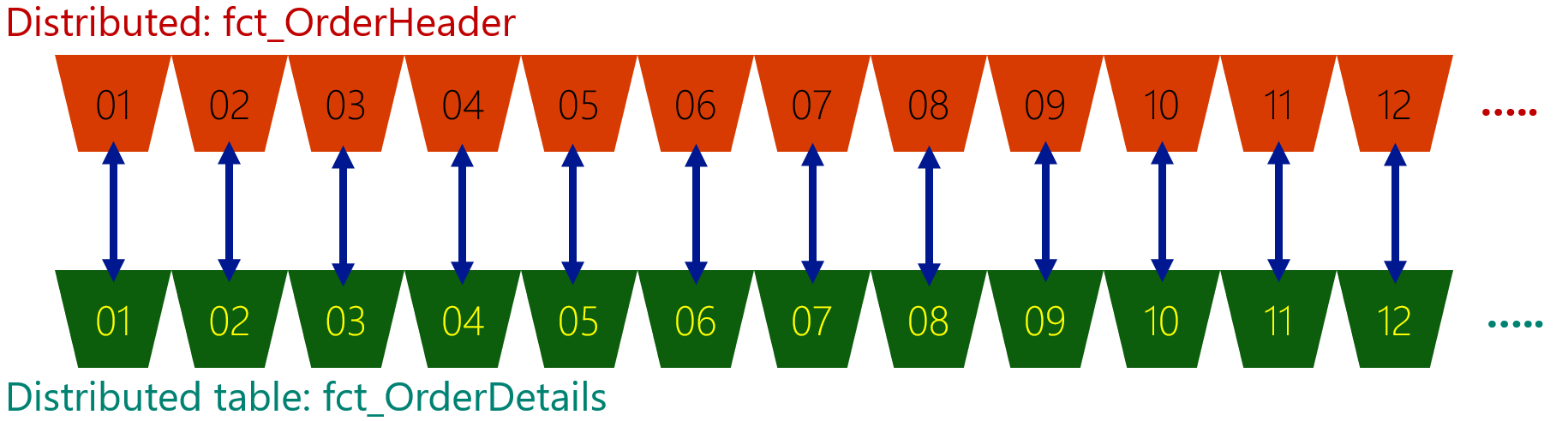
Data movement has the biggest performance impact of any single factor in SQL Pool DW. The MPP Query Optimizer uses a cost-based optimization, based on the Statistics for the tables involved in the query. A primary goal of the optimization is to reduce the amount of data (number of rows and data volume) that needs to be moved to satisfy the query.

## Why data moves

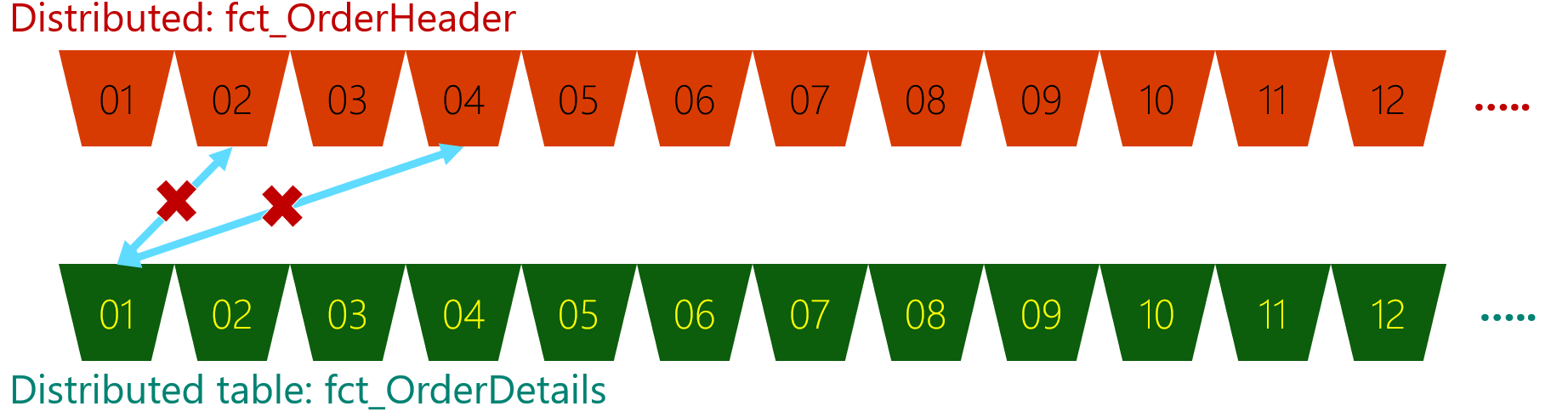
There are different scenarios that can incur data movement.

### Distribution Incompatible Join

When a query with a JOIN is processed against 2 or more Distributed tables, the query is refactored(\*) / sharded to execute the joins against each of the Distributions individually. The distributions are only joined to equivalent distributions in the joined table(s):



Cross-distribution joins are not allowed:



If a JOIN is executed against two tables that are distributed on key(s) that are not included in the JOIN / ON clause, 1 *or more* tables will need to be re-distributed to satisfy data co-location requirement

If the fct\_OrderHeader table is Distributed on OrderID, but fct\_OrderDetails is Distributed on ItemID, the data will not line up and a join on OrderID will not return any results unless data is moved to align, the fct\_OrderDetails table will be redistributed on OrderID.

If the fct\_OrderHeader table is Distributed on OrderDateTime, and fct\_OrderDetails is Distributed on ItemID, both tables will be redistributed on OrderID to satisfy a join on OrderID.

If the JOIN is on the Distribution key for all tables in the query, that is considered a Distribution Compatible Join, data should not move.

Note that all instances of the same key (such as OrderID) must be defined as the same data type. A Distribution key defined as INT is not distributed the same as the same key defined as BIGINT or VARCHAR(x). Hence, the same key, defined with different data types will imply Distribution Incompatible Joins, even if the key columns hold the same values.

(\*) all queries are always refactored by the SQL POOL DW Query Optimizer, even a single table SELECT statement.

### Distribution Incompatible Aggregation

When data is aggregated (such as COUNT\_BIG) on a column that is not the Distribution column, data will be moved so that the final aggregation can be computed.

Multiple columns in the GROUP BY will cause the query to be Distribution Incompatible by default – only 1 column can be the Hash Distribution key for a table.

The SQL POOL DW query optimizer will attempt to identify Distribution Compatible sub-aggregations that can be computed on each distribution, to reduce the amount of data to be moved. E.g. SELECT COUNT\_BIG(\*) … GROUP BY StoreID on a table Distributed on another key may execute the SELECT COUNT\_BIG(\*) … GROUP BY StoreID on each Distribution before moving data to the Control Node for final aggregation.

Note that Aggregation is very likely to incur data movement of some sort.

### Query Syntax

Different query syntaxes that are reasonable in regular SMP SQL can cause data movement in the SQL Pool DW, and hence slow processing of the query.

Some examples (but not all) are:

1. Functions in the predicate.

e.g. WHERE YEAR(OrderDate) = 2020

e.g. INNER JOIN a.Year = YEAR(b.OrderDate)

In these cases, the MPP Query optimizer can’t determine if the query is Distribution Compatible and may move data unnecessarily.

1. OVER() clauses.

Can cause data movement, if the Distribution key is not the first column listed in the OVER (PARTITION BY (…)) clause.

1. COUNT(DISTINCT()) clauses.

Can cause data movement, if the Distribution key is not the first column listed in the COUNT DISTINCT clause.

*Note that OVER() and COUNT(DISTINCT()) are expensive query patterns in SQL Pool DW, use with caution and perform Query Tuning as appropriate.*

1. OUTER JOIN

Can cause a Replicated table to be converted to a Distributed table. Always write SQL POOL DW queries as INNER JOINs if possible.

### Re-distribute Data

Any CTAS operation to a different table geometry, or to a different Distribution key:

* Replicated --> Distributed
* Distributed on colA --> Distributed on colB
* etc

### Round-Robin distributed tables

Any query that joins a RR table to anything other than a Replicated table is guaranteed to incur data movement. If there is any aggregation on a RR table query, that will also incur data movement.

## Data Movement types

Data movement is an essential and unavoidable query component in a SQL Pool DW environment. The goal of understanding the different types of data movement, and Performance Tuning is to reduce the number of rows / amount of data being moved as much as possible.

The SQL Pool DW query optimizer uses statistics to determine the cheapest series of (1 or more) move operations, that are required to complete the query.

These are the most common data movement operations, this is not a complete list of the different move types.

### Shuffle Move

This is the most common move seen in SQL Pool DW query plans. The Shuffle move will re-distribute a Distributed table on a new Distribution key. The source for the Shuffle move may be the whole table, or a filtered subset (by using a WHERE … clause in the DMS Read operation).

This move occurs because the tables are not joined on the Distribution key column for one or more tables, or the distribution keys are not compatible.

* A Shuffle move is used to redistribute a Hash distributed table on another column or when the key column has a different data type.
* A Shuffle Move is used to redistribute a Round Robin Distributed table as a Hash Distributed table.

### Broadcast Move

The Broadcast move will convert a Distributed table to a Replicated table. The source for the Broadcast move may be the whole table, or much more likely a filtered subset.

The Broadcast move is typically targeted at a relatively small data set, as a large Replicated table an be expensive to create on a high DWUc DW, due to the number of Replicated copies that are created.

* A Broadcast move is used to create a Replicated table from a Hash Distributed table.
* A Broadcast move is used to create a Replicated table from a Round Robin Distributed table.

### Partition Move

The Partition move will move an intermediate resultset to the Control node. This is typically done to allow for final aggregation before returning the results to the caller. However, it’s also possible that the resultset can be sent back to the Compute nodes in a Move operation.

* A Partition move coalesces data to the Control node whenever full-dataset processing is required, either to return to the client or to enable further processing in the query plan.

### Move

The Move operation will move a resultset from the Control node to create a Replicated table on the Compute nodes. This can be the result of an intermediate aggregation after a Partition Move, or as a result of the query referencing database metadata that is only available on the Control node, though the second scenario is not usually seen.

* A Move operation sends data from the Control node to the Compute nodes.

### 

### Trim Move

The Trim move will convert a Replicated table to a Distributed table. This scenario is somewhat counter-intuitive, as the Replicated table is already co-existent with all Distributions in the database, however, this is executed as part of the query plan for an Outer Join.

* A Trim move is used to create a Hash Distributed table from a Replicated table, in response to an OUTER JOIN.

## 

## Replicated tables

In the SQL POOL DW environment, a Replicated table is physically stored as a Round Robin Distributed table. On the first reference to a Replicated table(\*), the data will be fully read and copied to each Compute node as an in-memory Distributed table.

In this scenario, a different SQL Pool DW query plan may be generated.

(\*) any query, such as SELECT TOP 1 \* F ROM …, after any of the following:

1. Pause / Resume the DW
2. Resize the DW
3. Data modification to the table.