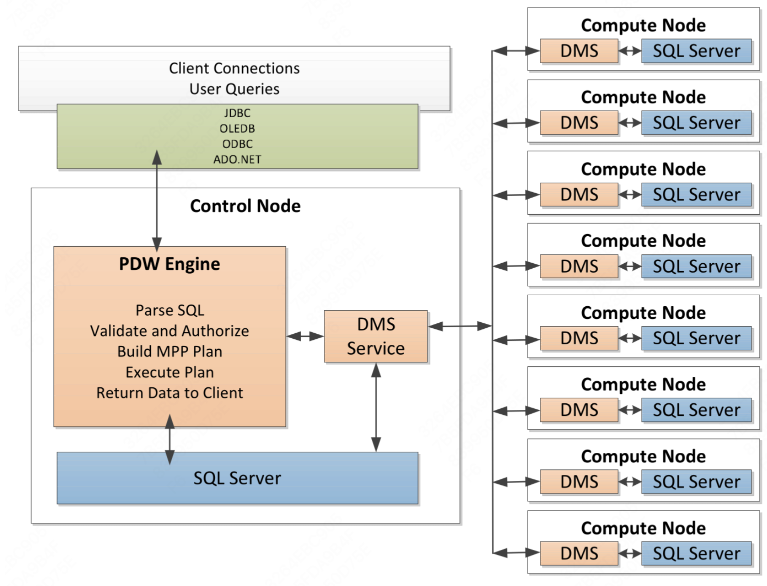
# Query Optimization in Microsoft SQL Server PDW

July 17, 2025

## 1 Introduction

### 1.1 Overview of SQL Server PDW

Microsoft SQL Server Parallel Data Warehouse is a shared-nothing parallel database appliance and is one example of an MPP system.



It has a **control node** that manages a number of compute nodes (see Figure 1.1). The control node provides the external interface to the appliance, and query requests flow through it. The control node is responsible for query parsing, creating a distributed execution plan, issuing plan steps to the compute nodes, tracking the execution steps of the plan, and assembling the individual pieces of the final results into the single result set that is returned to the user.

**Compute nodes** provide the data storage and the query processing backbone of the appliance. The control and compute nodes each have a single instance of SQL Server RDBMS running on them. User data is stored in tables that are hash-partitioned or replicate tables across the SQL Server instances on the compute nodes.

To execute a query, the control node transforms the user query into a distributed execution plan (called **DSQL plan**) that consists of a sequence of operations (called **DSQL Operations**). At a high-level, every DSQL plan is composed of two types of operations:

1. SQL operations, which are SQL statements to be executed against the underlying compute nodes' DBMS instances
2. DMS(Data Movement Service) operations which are operations to transfer data between DBMS instances on different nodes.

## 1.2 Query Optimization in PDW

Queries executed in MPP environments tend to be complex - involving many joins, nested sub-queries and aggregations - and are usually long-running and resource-intensive. The goal of the query optimizer is to find the best execution plan for a given query, which is usually accomplished by examining a large space of possible execution plans and comparing these plans according to their estimated execution costs.

PDW invokes the SQL Server QO against a "shell database" to obtain a compact representation of the **optimization search space** called a `MEMO`. This search space is then augmented with statistical information on the distribution of data in the appliance to find a parallel execution plan for the query, taking into consideration the available statistics and the actual data distribution in the appliance.

The main idea of PDW QO can be summarized as follows:

1. We store the metadata of the distributed tables in a "shell database" on a single SQL Server instance. The "shell database" provides the "single system image" of the data in the appliance. Importantly, it also stores aggregated statistical information on the user data.

2. Using the shell database, we use the existing compilation stack of SQL Server to parse a given query, and generate and export the space of execution alternatives (`MEMO`).

3. We traverse the space of execution alternatives to introduce data movement operations, and make a cost-based decision on the best execution plan for the distributed environment.

## 2 Microsoft SQL Server PDW Architecture Overview

### 2.1 Appliance

There are two distinct types of nodes that implement the query processing functionality

1. **Control Node**. The control node manages the distribution of query execution across the compute nodes, accepts client connections to the PDW appliance and manages client authentication. In addition to containing a SQL Server instance, the control node contains additional software to support the distributed architecture of the PDW. This includes the engine that coordinates the data warehousing functions that are specific to processing parallel queries, stores appliance-wide metadata and configuration data, and manages appliance and database authentication and authorization. The control node also manages the Data Movement Service (DMS) that runs on the appliance nodes and is the communication layer for transfering data between the nodes in the appliance.

2. **Compute Nodes**. Each compute node is the host for a single SQL Server instance. It also runs a DMS process for communication and data transfer with the other nodes in the appliance. Each compute node stores a portion of the user data.

Tables in a PDW appliance can either be

1. replicated on each compute node in the appliance, or

2. hash-partitioned on a specified column(s) across the compute nodes.

### 2.2 Shell Database

A "shell database" is a SQL Server database that defines all metadata and statistics about tables, but does not contain any user data.

The shell database also contains global statistics for all the tables in the appliance. To compute global statistics, local statistics are first computed on each node via the standard SQL Server mechanisms, and are then merged together to derive global statistics.

## 2.3 Data Movement

The Data Movement Service (DMS) is responsible for moving data between all the nodes on the appliance. Once instance of DMS runs on each of the control and compute nodes, certain steps of a user query may require intermediate result sets to be moved from one compute node to another. In addition, sometimes intermediate result sets from one or more compute nodes must be moved to the control node for final aggregations and sorting prior to returning the result set to the client. PDW utilizes temporary (*temp*) tables on the compute and control nodes as necessary to move data or store intermediate result sets. In some cases, queries can be written that generate no temp tables and results can be streamed from the compute nodes directly back to the client that issued the query - such queries will not involve DMS.

## 2.4 The DSQL Plan and its Execution

Given a user-specified SQL query, the PDW engine is responsible for creating a parallel execution plan (known as a DSQL plan). A DSQL plan may include the following types of operations:

- **SQL Operations** that are executed directly on the SQL Server DBMS instances on one or more compute nodes.
- **DMS Operations** which move data among the nodes in PDW for further processing, e.g. moving intermediate result sets from one compute node to another.
- **Temp table operations** that set up staging tables for further processing.
- **Return operations** which push data back to the client.

Query plans are executed serially, one step at a time. However, a single step typically involves parallel operations across multiple compute nodes.

**DSQL Plan Example**: Using the TPC-H schema as an example, let's assume that the `Customer` table is hash-partitioned on `c_custkey`, and the

Orders table is hash-partitioned on o_orderkey and we want to perform the following join between these two tables.

```sql
SELECT c_custkey,
       o_orderdate
FROM Orders, Customer
WHERE o_custkey = c_custkey AND o_totalprice > 100
```

The table partitioning is not compatible with the join since Orders is not partitioned on o_custkey. Thus, a data movement operation is required in order to evaluate the query. The optimizer on the control node may produce a DSQL plan consisting of the following two steps:

1. DMS Operation that repartitions data in the Orders table on o_custkey in preparation for the join.

2. Return SQL Operation that selects tuples for the final result set from each compute node and returns them back to the client.

**Step 1: DMS Operation**: In the example above, the first step in the DSQL plan is a DMS operation that repartitions Orders data on o_custkey. The DMS operation specifies the How is repartition done

1. SQL statement required to extract the source data

2. the tuple routing policy (e.g., replicate or hash-partition on a particular column), and

3. the name of a (temporary) destination table.

The Engine service then begins broadcasting the DMS operation from the control node to the DMS instance on each node. Upon receiving the DMS message, the DMS instance on each compute node begins execution of the data movement operation by issuing the SQL statement below:

```sql
SELECT o_custkey,
       o_orderdate
FROM Orders
WHERE o_totalprice > 100
```

against the local SQL Server instance. Each DMS instance reads the result tuples out of the local SQL Server instance, routes the tuples to the appropriate DMS process by hashing on o_custkey, and also inserts the tuples it receives from other DMS instances into the specified local destination table (`Temp_Table` in this example). Once all of the tuples from the source SQL statement have been inserted into their respective destinations the DMS operation is complete.

**Step 2: SQL Operation**: After the DMS operation has completed, the Engine service moves on to the second step in the plan, which is the SQL operation that is used to pull the result tuples from each compute node. To perform this operation, the Engine service obtains a connection to the SQL Server instance on each compute node and issues a specified SQL statement. In this case, the SQL statement that will be executed is:

```
SELECT c.c_custkey,
       tmp.o_orderdate
FROM Customer c,
     Temp_Table tmp
WHERE c.c_custkey = tmp.o_custkey
```

## 2.5 Cost-Based Query Optimization PDW

Figure 2.5 provides the high-level data flow for PDW query optimization. The key observation that forms the basis of PDW QO is that the problem of

1. algebrizing input queries into operator trees, and
2. the logical (as opposed to physical, or partition-dependent) exploration done on operator trees to find plan alternatives is the same for PDW as it is for a single SQL server instance.

1. **PDW Parser**: This component is responsible for parsing the input query string and creating an abstract syntax tree (AST) structure that can be validated against PDW syntax rules. Some PDW queries may also need a few basic transformations before they are ready to be sent to SQL Server against the shell database.
2. **SQL Server Compilation**: After validation by the PDW parser, the query is passed to SQL Server for compilation against the shell database. The SQL Server optimizer performs the following functions:

   (a) Simplification of the input operator tree into a normalized form. This is inserted as the initial plan into the `MEMO` data structure, which will hold the space of alternative plans for the query.
   (b) Logical transformations on the plans in the `MEMO` data structure to augment the set of choices. These are based on relational algebra rules. For instance, all equivalent join orders are generated in this stage.
   (c) Estimation of the size of intermediate results for each of the execution alternatives. These estimations are based on the size of base tables and statistics on the column values.
   (d) The implementation phase which adds physical operator (algorithms) choices into the search space. The optimizer costs them and prunes the plans that do not meet established lower bounds.
   (e) Extraction of the optimal execution plan.

3. **XML Generator**: This component takes the search space generated by SQL Server optimizer represented in the MEMO data structure as its input and encodes the information as XML.

4. **PDW Query Optimizer**: The PDW query optimizer is the consumer of the search space output from the XML generator. There is a memo parser on the PDW side which is re- sponsible for constructing the memo data structure for the PDW query optimizer. Once the memo data structure for the PDW side is constructed, the PDW optimizer performs bottom-up optimization with the help of the PDW cost model. The responsibilities of the PDW query optimizer include:

   (a) Enumeration of distributed execution plans by systematically adding appropriate data movement strategies into the search space.
   (b) Costing the alternative plans generated using the PDW cost model.
   (c) Choosing the optimal (minimal cost) distributed tion plan

**Example 2.1.** Consider

```sql
SELECT *
FROM CUSTOMER C, ORDERS O
WHERE C.C_CUSTKEY = O.O_CUSTKEY
AND O.O_TOTALPRICE > 1000
```

## 3 Problems

## 4 References