Resolving Queries

Understanding Data Movement

Agenda

- Understanding D-SQL Plans
- Understanding Data Movement
- Data Movement Service (DMS)
- Optimising DMS

Understanding D-SQL plans

Accessing the D-SQL plan

Before execution

- Use EXPLAIN

After execution

- DMV: sys.dm_pdw_request_steps
- DMV: sys.dm_pdw_sql_requests
- Management Console: Queries windows

EXPLAIN

- PDW Specific syntax
- Returns the D-SQL plan (XML)
- Like an estimated execution plan but for MPP
- Sometimes called the MPP Plan
- Details the operations or steps required to resolve the query

Example usage

```
EXPLAIN
SELECT p.*
FROM dbo.FactInternetSales fis
LEFT JOIN dbo.DimCustomer p on fis.CustomerKey = p.CustomerKey
WHERE fis.OrderDateKey = 20040101
OPTION (LABEL = 'Shuffle : Join Cost')
;

EXPLAIN
CREATE TABLE dbo.DimEmployee
WITH (DISTRIBUTION = Hash(EmployeeKey))
AS
SELECT*
FROM[AdventureWorksPDW2012].dbo.DimEmployee
OPTION (LABEL = 'Trim')
;
```

Usage Guidance

Explain can be used before

- Select
- Insert
- Update
- Delete
- CTAS
- CRTAS
- CETAS

Explain cannot be used in conjunction with

- Variables
- Stored Procedures
- DDL

D-SQL Operations

SQL Operations

- Rnd_ID
- On
- Return

- RemoteOn
- MetadataCreate

DMS Operations

- Shuffle
- Broadcast
- Partition
- Move
- Trim
- DistributeReplicated
- Copy

SQL Operations

Rnd_ID / Random ID

- Used to create unique names
- Most common usage : create names for temporary tables used in data movement

On

- Used to perform an action on a database or an object
- Typical use case is to create a temporary table for data movement

Return

  - Used to return final result set back to end user
  - Max of one return step in any MPP plan
  - Return may also be used to perform the final aggregation

DMS Operations

- Data Movement Service implements DMS operations
- Equate to data movement strategies
- Transparent to the user
- Optional steps – data doesn’t *have* to move
- Fulfil data movement requests

Typical Orchestration Steps

1. Random ID Operation
   - Create random name for temporary table
2. ON Operation
   - Create Temporary table using randomly generated name
3. DMS Operation
   - Move data into position to satisfy query request
4. Return Operation
   - Return results back to user
5. ON Operation
   - Drop Temporary Table

Understanding Data Movement

Why Data Moves

- Incompatible Join
- Incompatible Aggregation
- Re-distribute data
- Data Consistency
- Query syntax

Joining Data

- Data is spread across an appliance
- Before a join can take place data needs to be co-located
- One or more sets of data may need to be re-distributed to enable the join
- Not all joins require data re-distribution

Join Compatibility Matrix

Left Table	Right Table	Inner	Left	Right	Full	Cross
Replicated	Replicated	🟩	🟩	🟩	🟩	🟩
Replicated	Distributed	🟩	🟥	🟩	🟥	🟩
Distributed	Replicated	🟩	🟩	🟥	🟥	🟩
Distributed	Distributed	🟧	🟧	🟧	🟧	🟥

Conditions!

For a Distributed – Distributed join to be compatible (green) join must

- Contain distribution key of both columns
- Match data types on distribution keys
- Be an equality join

Incompatible Join Example

```
EXPLAIN
SELECT    p.*
FROM      dbo.FactInternetSales fis
LEFT JOIN dbo.DimCustomer p
ON        fis.CustomerKey = p.CustomerKey
WHERE     fis.OrderDateKey = 20040101
OPTION    (LABEL = 'Shuffle : Join Cost');
```

- FactInternetSales distributed on OrderDateKey
- DimCustomer is distributed on Customer
- Therefore join incompatible
- Resolution: Move FactInternetSales. DimCustomer is distributed on the joining key
- N.B. Filter is applied to FactInternetSales prior to the move

Aggregation Incompatibility

Data is aggregation incompatible when

- Data needs to be moved for a full aggregation to take place

Two approaches to resolve the incompatibility:

- Re-distribute data by a column in the group by
  - Keeps data down on the compute nodes
- Push data to a central point for aggregation
  - Uses the control node

Incompatible Aggregation example

```
EXPLAIN
SELECT COUNT(*)
FROM dbo.FactInternetSales fis
GROUP BY ProductKey
OPTION (LABEL = 'Shuffle : Aggregate');
```

- FactInternetSales distributed by OrderDateKey
- Query groups by product
- Therefore aggregation incompatible
- Resolution: Move FactInternetSales and re-distribute data on ProductKey
- N.B. Data is pre-aggregated by ProductKey prior to movement

Re-distributing Data

Typically found when data is being persisted rather than returned to the user

You can move:

- From distributed to replicated
- From replicated to distributed
- From distributed (a) to distributed (b)

Re-distribution Example

```
EXPLAIN
CREATE TABLE dbo.DimEmployee_dist
WITH (DISTRIBUTION = Hash(EmployeeKey))
AS
SELECT*
FROM[AdventureWorksPDW2012].dbo.DimEmployee
OPTION (LABEL = 'Trim')
;
```

Distributed

Replicated

- DimEmployee is a replicated table
- CTAS requests create DimEmployee_dist hashed on EmployeeKey
- Therefore data needs to be re-hashed by Employee key
- Data also needs to be persisted in the new table
- N.B. Only data for this compute node needs to be kept

Query Syntax

Depending on the distribution key

- OVER clause
- DISTINCT counts

… may trigger data movement…

Any expression on the distribution key will also trigger data movement

What Data Moves

As little as is possible!

- Remove columns
  - Retain columns required for query resolution
- Remove rows
  - Apply where clause predicates
- Pre-aggregate Data
  - Group by the distribution key for partial aggregation
- Transport remote rows
  - Only send rows to other nodes that need to be stored remotely

How Data Moves

Via Data Movement Service (DMS)

- Component of PDW
- Exists on Control and Compute nodes
- Responsible for all load & query data movement
- Controlled by the PDW Engine (DMSManager.dll)

Data Movement Service

DMS Functionality

Primary functionality

- Re-distribute data across the compute nodes
- Centralize data to the control node
- De-centralizes data from control to compute
- Import and export data
  - Polybase
  - Remote Tables (export only)
- Load data

DMS Components

- DMS Manager
- DMS Core Service

DMS Manager

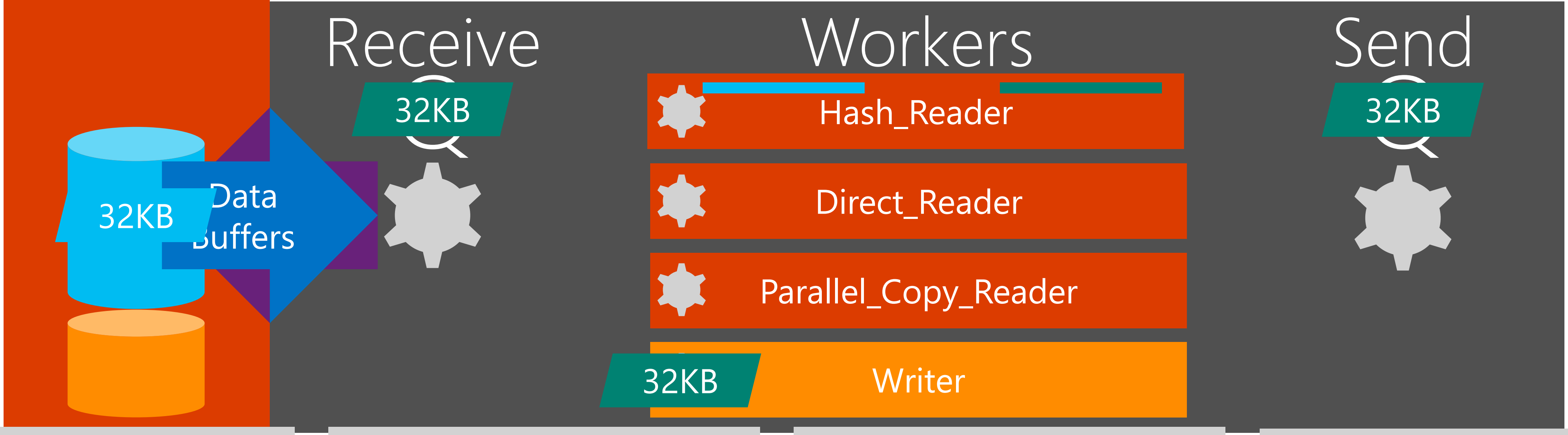
- Part of the PDWEngine service
- Exists only on the control node
- Takes instruction from the PDW Engine
- Communicates instructions to the DMS Core Services

DMS Core Service

- Runs on control node and all compute nodes
- Receives instructions from the DMS manager
- Responsible for physically moving data
  - Reads the data from the tables in PDW
  - Moves the data to the new location
  - Writes the data to the target

DMS Concepts

- Buffers
- Data Channel
- Queues
- Workers

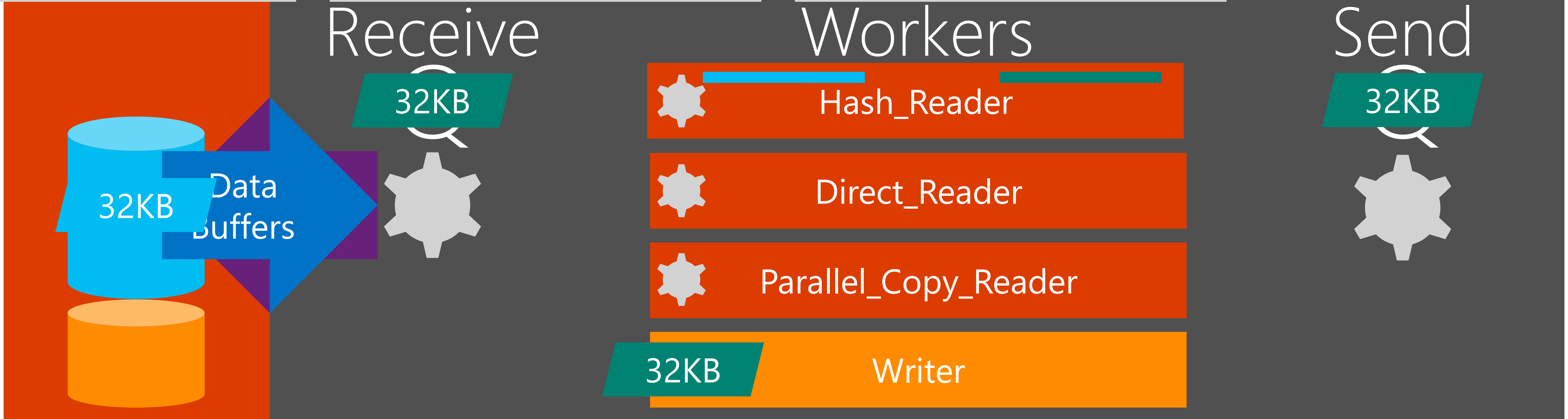


Step 1.
Execute query &
receive buffers

Step 2.
DMS Reader pulls
buffer & processes

Step 3.
Sender ships new
buffer to target node

Step 4.
Writer bulk loads
data into target table



DMS Buffers

- Transports data between nodes using buffers
- Buffers are capped in size and are formatted to native ODBC types
- When loading DMS uses 256KB buffers to ingest data
- Majority of the time DMS uses 32KB buffers
- A row must be able to fit inside a 32KB buffer

Data Channels

- DMS transports buffers over data channels
- Data channels instantiated when PDW started
- Shared channels are created between each of the DMS Core processes
- Binary & Reject Channels are created for load
- Default data channel for movement
- Sender threads transmit buffers
- Receiver threads receive buffers delivering them to appropriate *workers*

Example

- 6 Compute Nodes = 7 DMS Core Services
- 6x7 = 42 connections in each channel

Queues

Each DMS Core Service operates two queues

- Receive Queue
- Send Queue

Why? De-coupling

- Reader worker tasks de-coupled from data channel sender tasks
- Data channel receiver tasks de-coupled from the writer worker tasks

Workers

- Hash_Reader
  - Used by Shuffle, Trim
- Direct_Reader
  - Used by Broadcast, Move, DistributeReplicated
- Parallel_Copy_Reader*
  - Used by Partition, Remote Table Copy
- Writer
  - Used for all writes

Reader Workers

- Acquires data from the source
- Process the data inside the buffer row by row
- Hash the distribution key if required
- Write the row to the appropriate send buffer

Writer Workers

- One writer worker per target table
- If the table is a heap then write can also use BU locks and invoke multiple threads
- Takes work from the appropriate Receive Queue
- Invokes ODBC Bulk API to load data into target

Data Type Handling

- Pre-dominantly handled via SQL Native Client
  - ODBC and ODBC BCP api
  - Fast with minimal type conversion
- Some legacy code paths use ADO.NET
  - Results in Native to managed datatype overheads
  - Slower with greater overheads
  - Used by SSIS destination adaptor

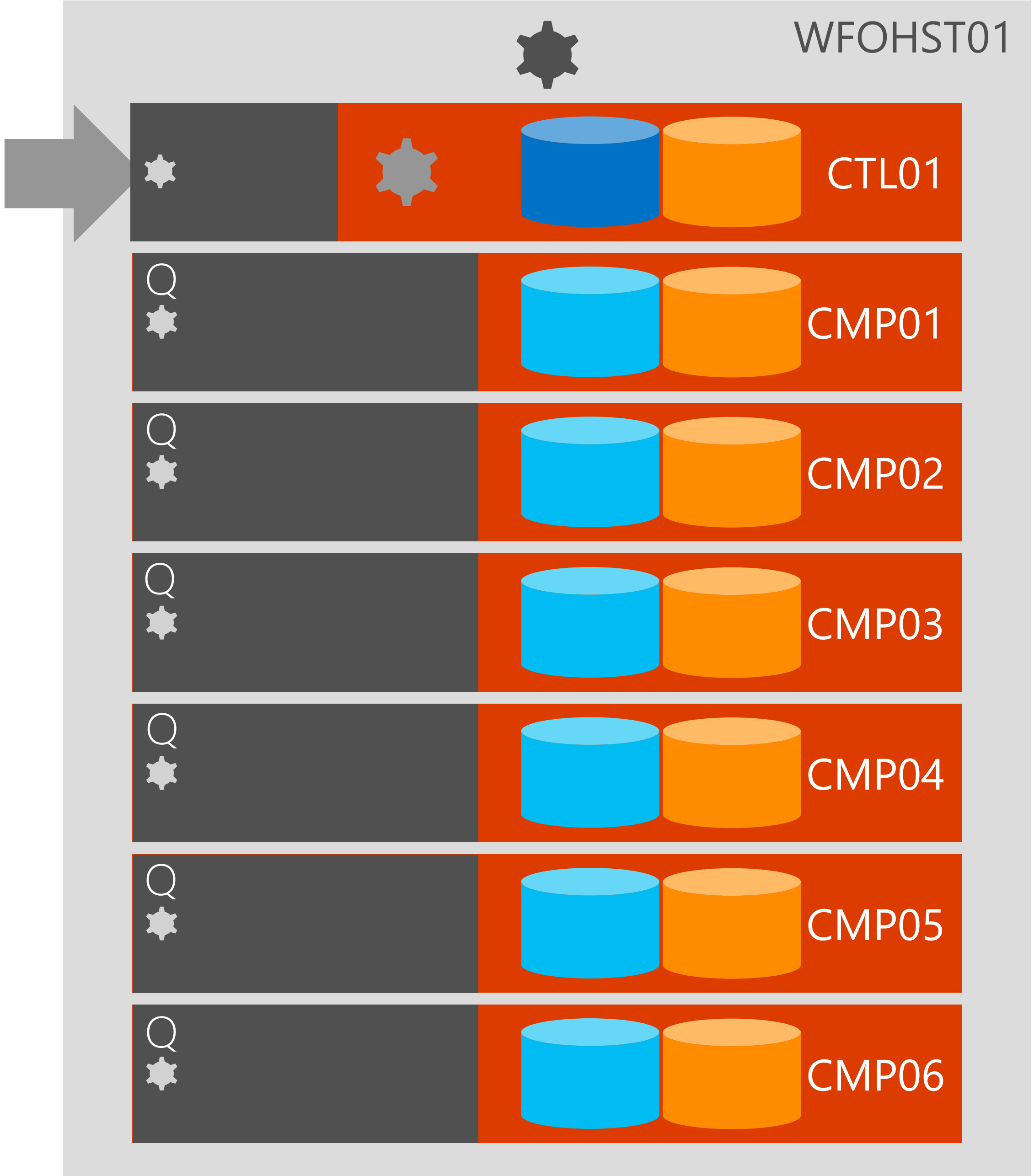
DMS Operations

DMS moves data via the following operations

- Shuffle
- Broadcast
- Partition Move
- Master Move
- Trim
- DistributeReplicated

Key

- Tempdb
- Shell database
- Compute database
- PDW Engine Service
- DMS Service
- PDW Cluster

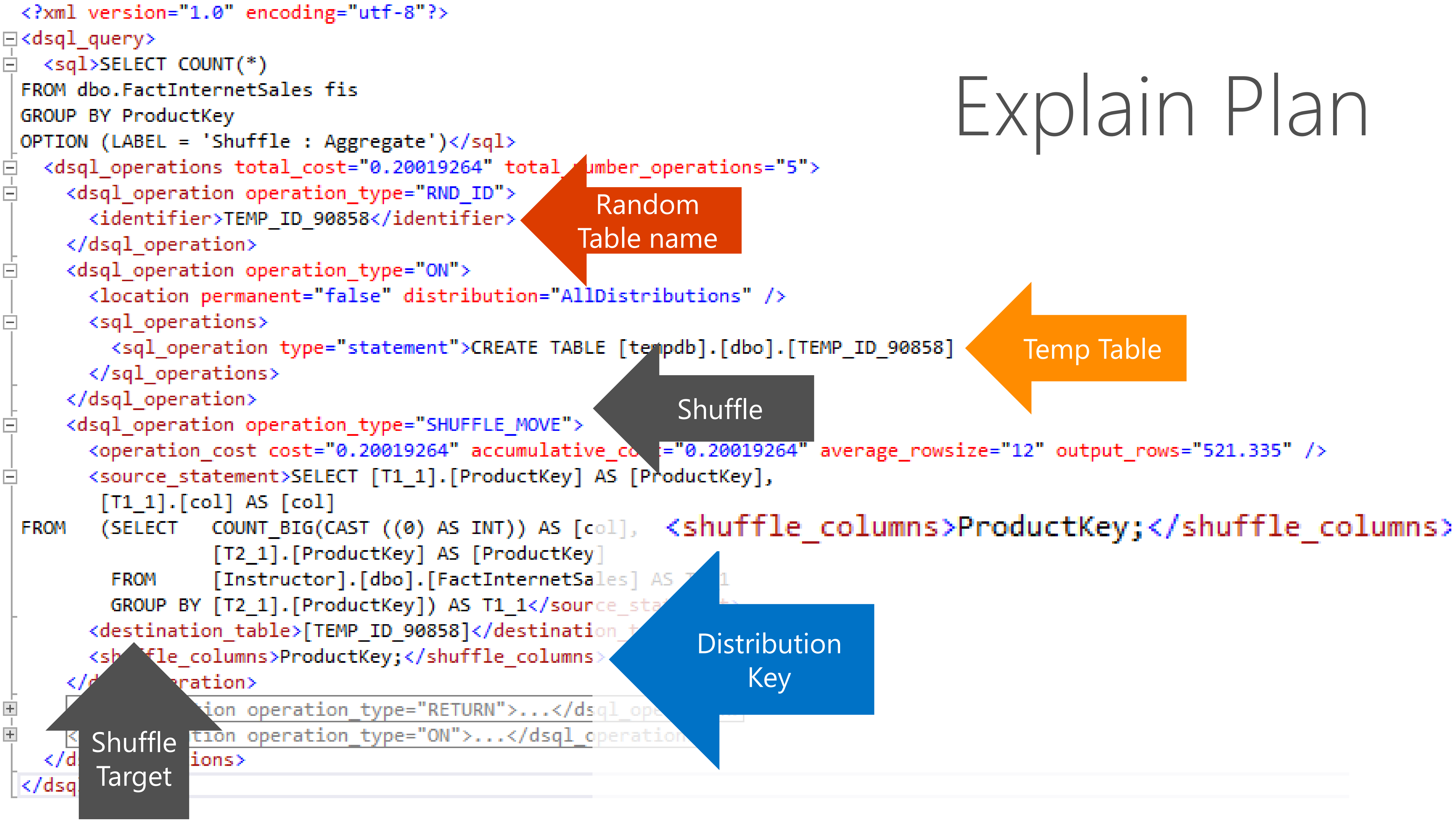


Shuffle Move Scenario

Introducing the Shuffle

- Re-distribute data on a different key than the current distribution
- Where possible data will be filtered and aggregated prior to movement
- Join must equal true for a Shuffle to be used
  - Cannot have A=B OR B=C

Explain Plan



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>SELECT COUNT(*)
FROM dbo.FactInternetSales fis
GROUP BY ProductKey
OPTION (LABEL = 'Shuffle : Aggregate')</sql>
  <dsql_operations total_cost="0.20019264" total_number_operations="5">
    <dsql_operation operation_type="RND_ID">
      <identifier>TEMP_ID_90858</identifier>
    </dsql_operation>
    <dsql_operation operation_type="ON">
      <location permanent="false" distribution="AllDistributions" />
      <sql_operations>
        <sql_operation type="statement">CREATE TABLE [tempdb].[dbo].[TEMP_ID_90858]
      </sql_operations>
    </dsql_operation>
    <dsql_operation operation_type="SHUFFLE_MOVE">
      <operation_cost cost="0.20019264" accumulative_cost="0.20019264" average_rowsize="12" output_rows="521.335" />
      <source_statement>SELECT [T1_1].[ProductKey] AS [ProductKey],
       [T1_1].[col] AS [col]
FROM   (SELECT   COUNT_BIG(CAST ((0) AS INT)) AS [col],
                 [T2_1].[ProductKey] AS [ProductKey]
        FROM     [Instructor].[dbo].[FactInternetSales] AS T2_1
        GROUP BY [T2_1].[ProductKey]) AS T1_1</source_statement>
      <destination_table>[TEMP_ID_90858]</destination_table>
      <shuffle_columns>ProductKey;</shuffle_columns>
    </dsql_operation>
    <dsql_operation operation_type="RETURN">...</dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
  </dsql_operations>
</dsql_query>
```

```
<shuffle_columns>ProductKey;</shuffle_columns>
```

Random Table name

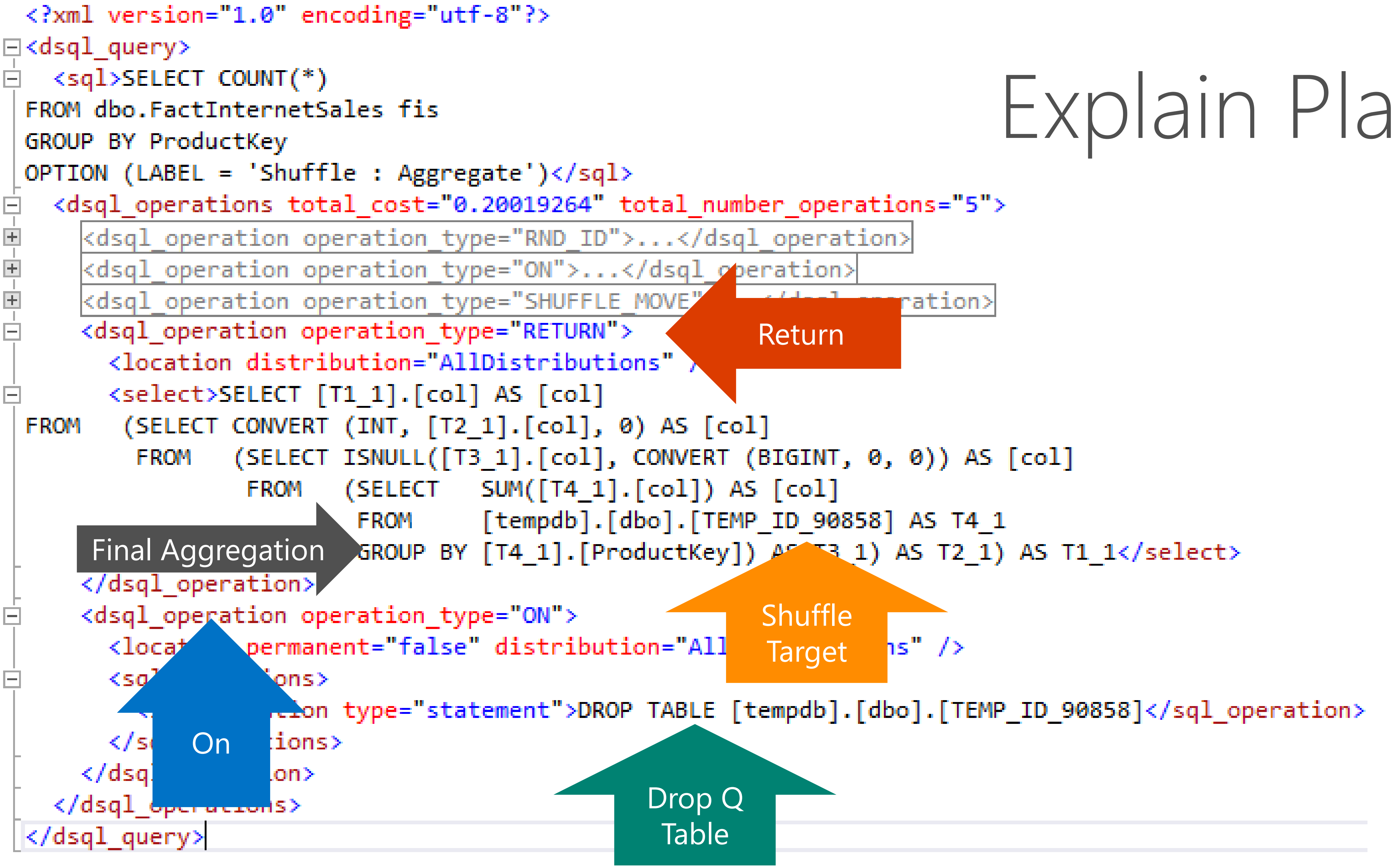
Temp Table

Shuffle

Distribution Key

Shuffle Target

Explain Plan



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>SELECT COUNT(*)
FROM dbo.FactInternetSales fis
GROUP BY ProductKey
OPTION (LABEL = 'Shuffle : Aggregate')</sql>
  <dsql_operations total_cost="0.20019264" total_number_operations="5">
    <dsql_operation operation_type="RND_ID">...</dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="SHUFFLE_MOVE">...</dsql_operation>
    <dsql_operation operation_type="RETURN">
      <location distribution="AllDistributions" />
      <select>SELECT [T1_1].[col] AS [col]
FROM   (SELECT CONVERT (INT, [T2_1].[col], 0) AS [col]
        FROM   (SELECT ISNULL([T3_1].[col], CONVERT (BIGINT, 0, 0)) AS [col]
                FROM   (SELECT   SUM([T4_1].[col]) AS [col]
                        FROM     [tempdb].[dbo].[TEMP_ID_90858] AS T4_1
                        GROUP BY [T4_1].[ProductKey]) AS T3_1) AS T2_1) AS T1_1</select>
    </dsql_operation>
    <dsql_operation operation_type="ON">
      <location permanent="false" distribution="AllDistributions" />
      <sql_operations>
        <sql_operation type="statement">DROP TABLE [tempdb].[dbo].[TEMP_ID_90858]</sql_operation>
      </sql_operations>
    </dsql_operation>
  </dsql_operations>
</dsql_query>
```

Return

Final Aggregation

Shuffle Target

On

Drop Q Table

Shuffle – Part 1

User submits query

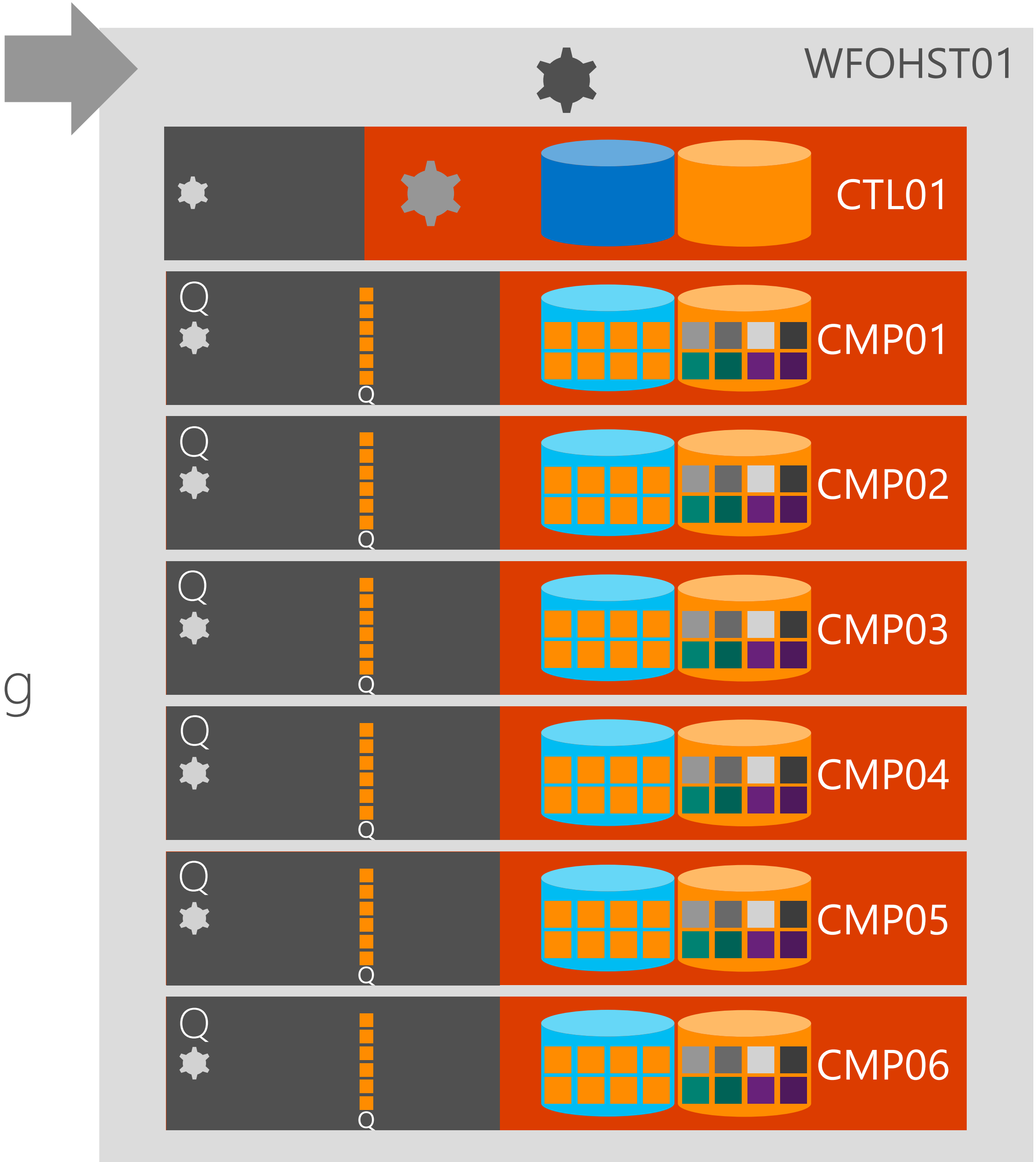
PDWEngine

- Creates a DSQL plan, by querying the shell database
- Generates Random ID
- Creates geometry aligned temp tables
- Instructs DMS to read data by executing a dsql query against the distributions

```
SELECT COUNT(*)
FROM dbo.FactInternetSales
GROUP BY ProductKey;
```

DMS

- Executes query from PDWEngine
- Query filters and pre-aggregates data
- Receives data in 32KB buffers
- Places buffers in a read queue for processing



Shuffle – Part 2

The DMS Hash_Reader Worker

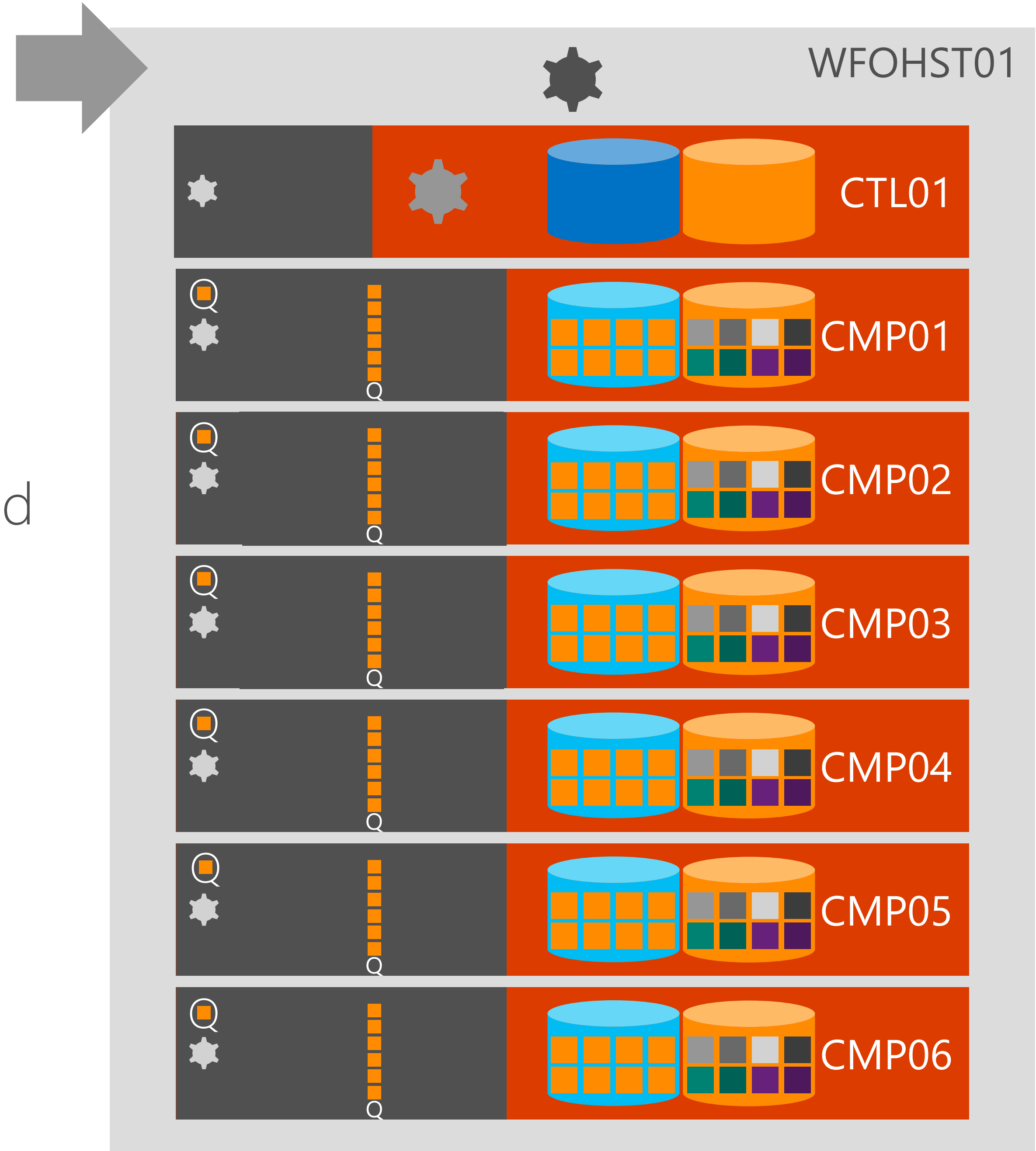
- Takes buffers from the read queue
- Read the buffer row by row
- Hash new distribution key column
- Write row into new distribution aligned buffer

Sender

- Transmits full buffers to target distribution

Receiver

- Places buffer on distribution aligned write queue



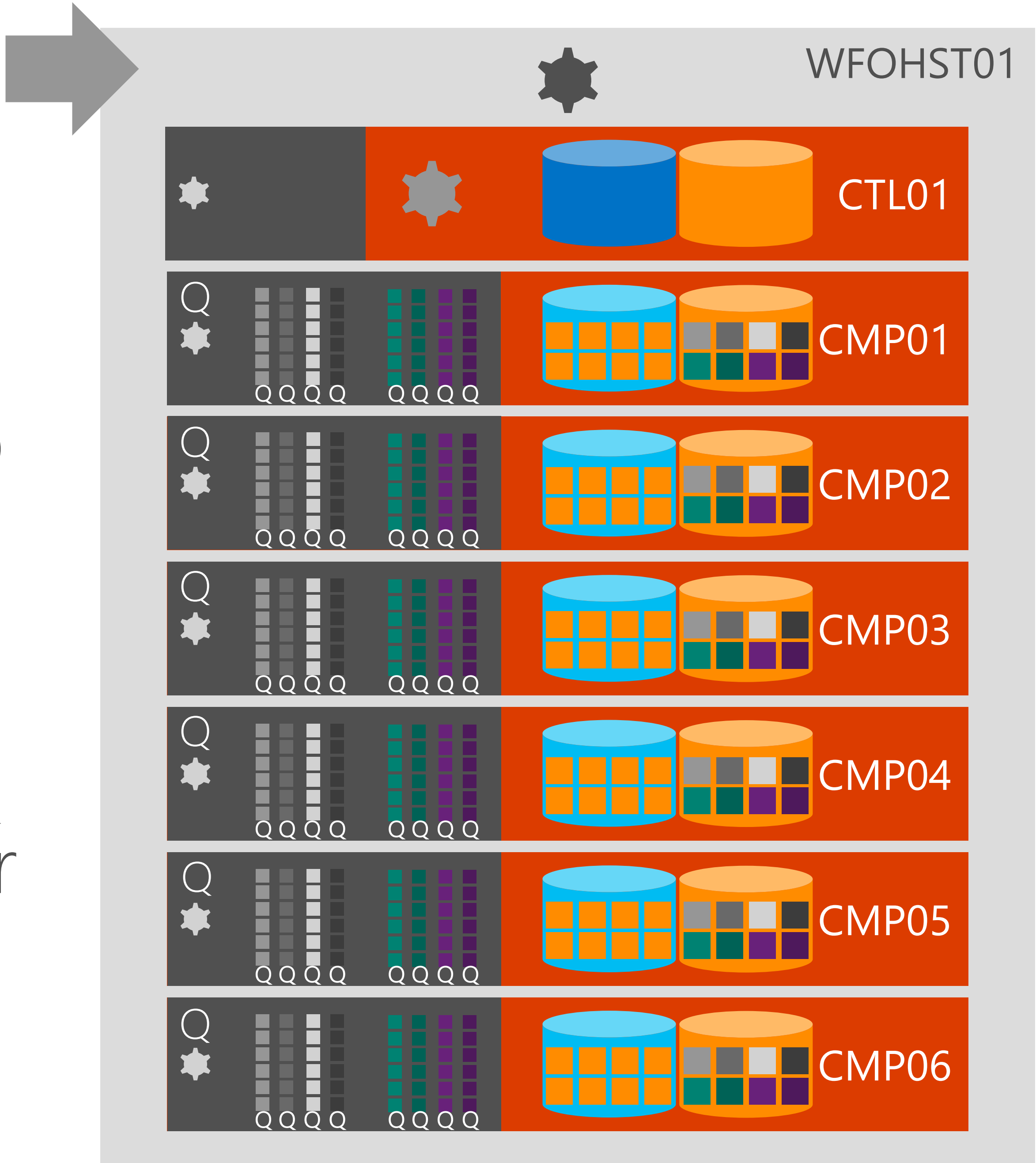
Shuffle – Part 3

DMS Writer Worker

- Bulk load buffers into distribution aligned Tempdb “Q” tables

PDWEngine

- Submits distribution compatible query against Q tables returning data to user
- Drops Q Tables

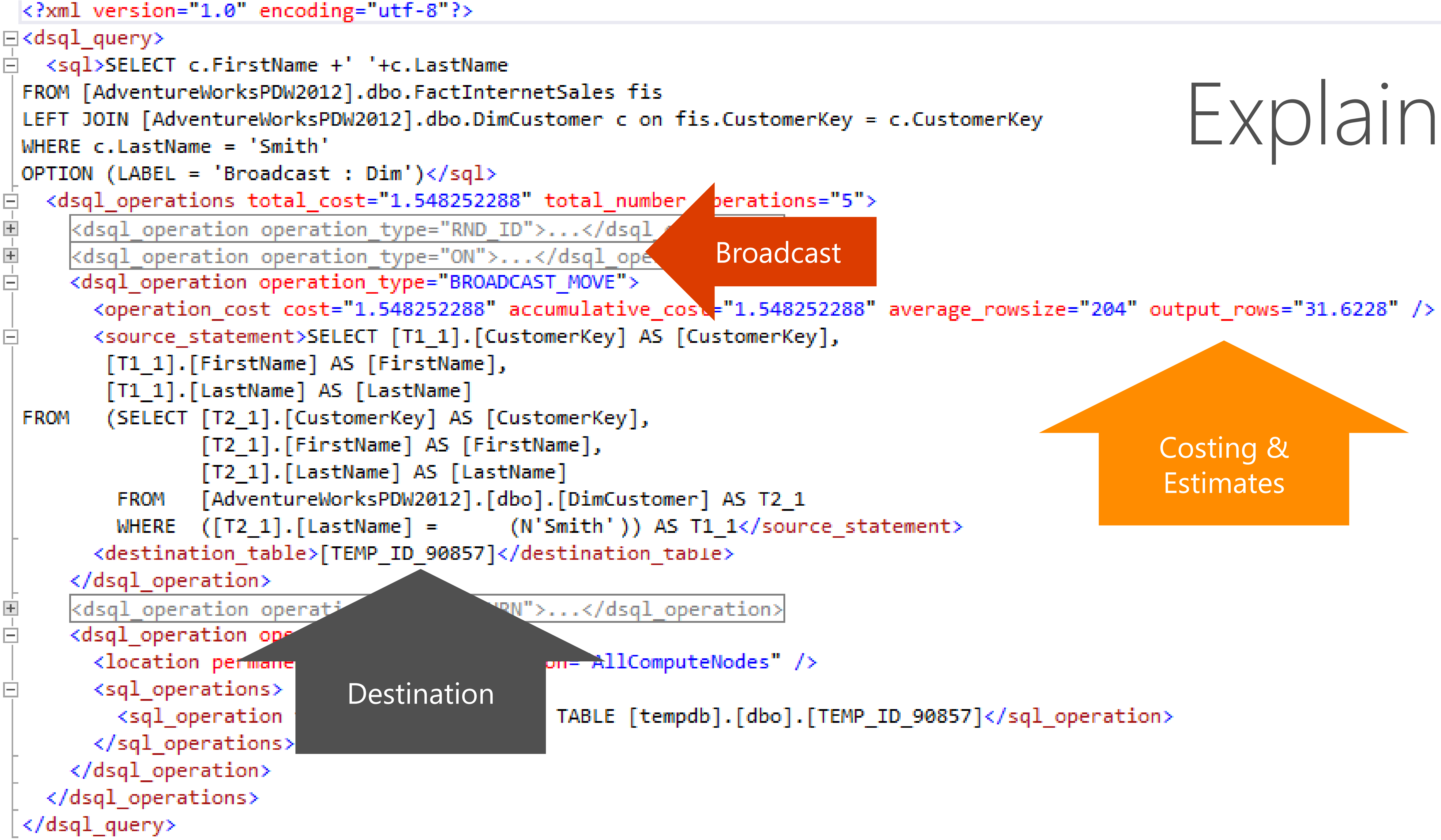


Broadcast Move Scenario

Understanding Broadcast

- Distributed data > Replicated data
- Copies qualifying data from distributions and replicates it to all nodes
- Typically used when the cost estimation for the movement is low
- No hashing of the data is required
- Worker type is DIRECT_READER
- Broadcast is the failsafe movement type

Explain

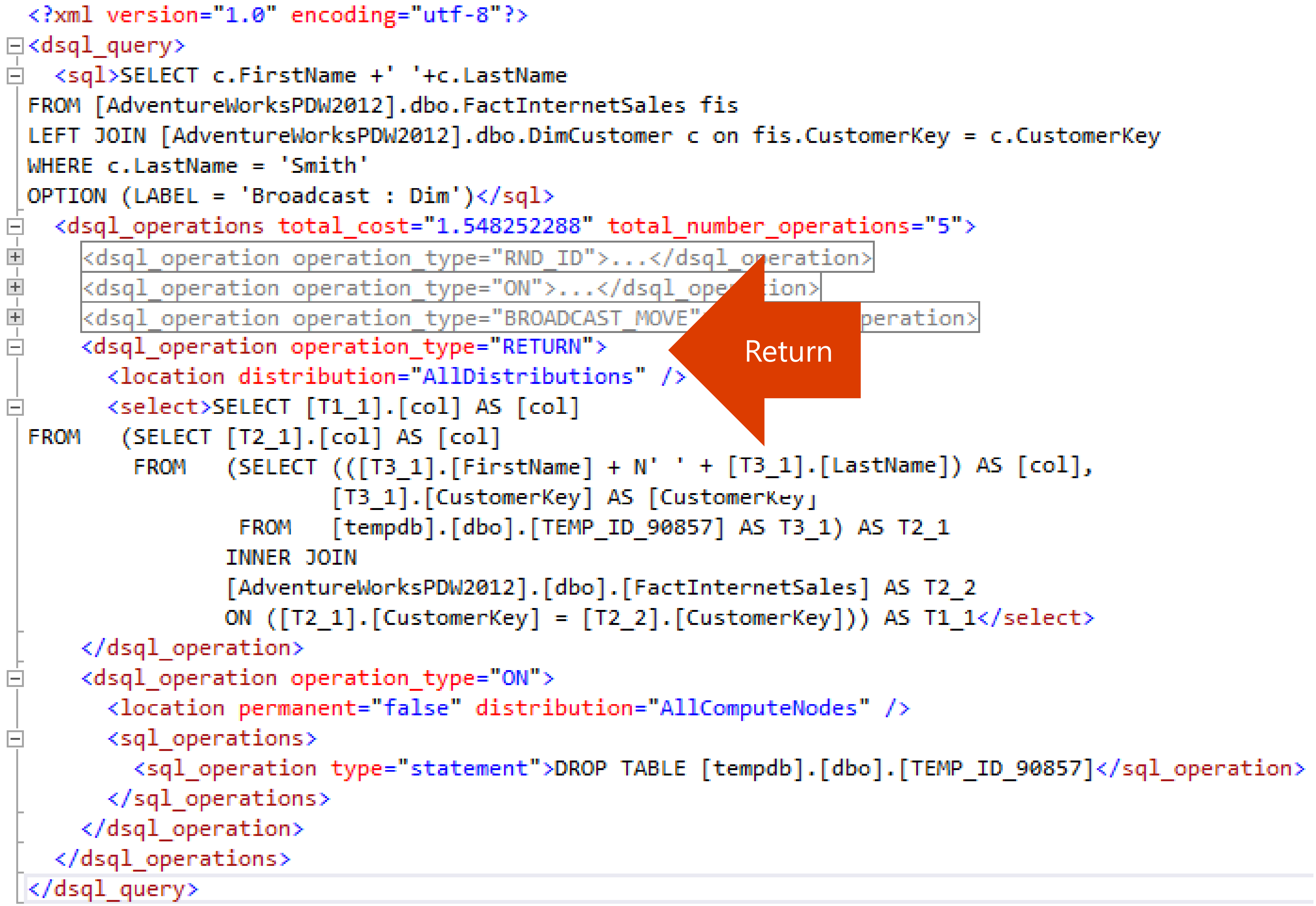


```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>SELECT c.FirstName +' '+c.LastName
FROM [AdventureWorksPDW2012].dbo.FactInternetSales fis
LEFT JOIN [AdventureWorksPDW2012].dbo.DimCustomer c on fis.CustomerKey = c.CustomerKey
WHERE c.LastName = 'Smith'
OPTION (LABEL = 'Broadcast : Dim')</sql>
  <dsql_operations total_cost="1.548252288" total_number   perations="5">
    <dsql_operation operation_type="RND_ID">...</dsql_
    <dsql_operation operation_type="ON">...</dsql_ope
    <dsql_operation operation_type="BROADCAST_MOVE">
      <operation_cost cost="1.548252288" accumulative_cos ="1.548252288" average_rowsize="204" output_rows="31.6228" />
      <source_statement>SELECT [T1_1].[CustomerKey] AS [CustomerKey],
       [T1_1].[FirstName] AS [FirstName],
       [T1_1].[LastName] AS [LastName]
FROM   (SELECT [T2_1].[CustomerKey] AS [CustomerKey],
               [T2_1].[FirstName] AS [FirstName],
               [T2_1].[LastName] AS [LastName]
        FROM   [AdventureWorksPDW2012].[dbo].[DimCustomer] AS T2_1
        WHERE  ([T2_1].[LastName] =      (N'Smith')) AS T1_1</source_statement>
      <destination_table>[TEMP_ID_90857]</destination_table>
    </dsql_operation>
    <dsql_operation operati              RN">...</dsql_operation>
    <dsql_operation op
      <location per                   on="AllComputeNodes" />
      <sql_operations>
        <sql_operation                    TABLE [tempdb].[dbo].[TEMP_ID_90857]</sql_operation>
      </sql_operations>
    </dsql_operation>
  </dsql_operations>
</dsql_query>
```

Broadcast

Costing & Estimates

Destination



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>SELECT c.FirstName +' '+c.LastName
FROM [AdventureWorksPDW2012].dbo.FactInternetSales fis
LEFT JOIN [AdventureWorksPDW2012].dbo.DimCustomer c on fis.CustomerKey = c.CustomerKey
WHERE c.LastName = 'Smith'
OPTION (LABEL = 'Broadcast : Dim')</sql>
  <dsql_operations total_cost="1.548252288" total_number_operations="5">
    <dsql_operation operation_type="RND_ID">...</dsql_o eration>
    <dsql_operation operation_type="ON">...</dsql_ope   tion>
    <dsql_operation operation_type="BROADCAST_MOVE"       peration>
    <dsql_operation operation_type="RETURN">
      <location distribution="AllDistributions" />
      <select>SELECT [T1_1].[col] AS [col]
FROM   (SELECT [T2_1].[col] AS [col]
        FROM   (SELECT (([T3_1].[FirstName] + N' ' + [T3_1].[LastName]) AS [col],
                        [T3_1].[CustomerKey] AS [CustomerKey]
                 FROM   [tempdb].[dbo].[TEMP_ID_90857] AS T3_1) AS T2_1
               INNER JOIN
               [AdventureWorksPDW2012].[dbo].[FactInternetSales] AS T2_2
               ON ([T2_1].[CustomerKey] = [T2_2].[CustomerKey])) AS T1_1</select>
    </dsql_operation>
    <dsql_operation operation_type="ON">
      <location permanent="false" distribution="AllComputeNodes" />
      <sql_operations>
        <sql_operation type="statement">DROP TABLE [tempdb].[dbo].[TEMP_ID_90857]</sql_operation>
      </sql_operations>
    </dsql_operation>
  </dsql_operations>
</dsql_query>
```

Broadcast – Part 1

User submits query

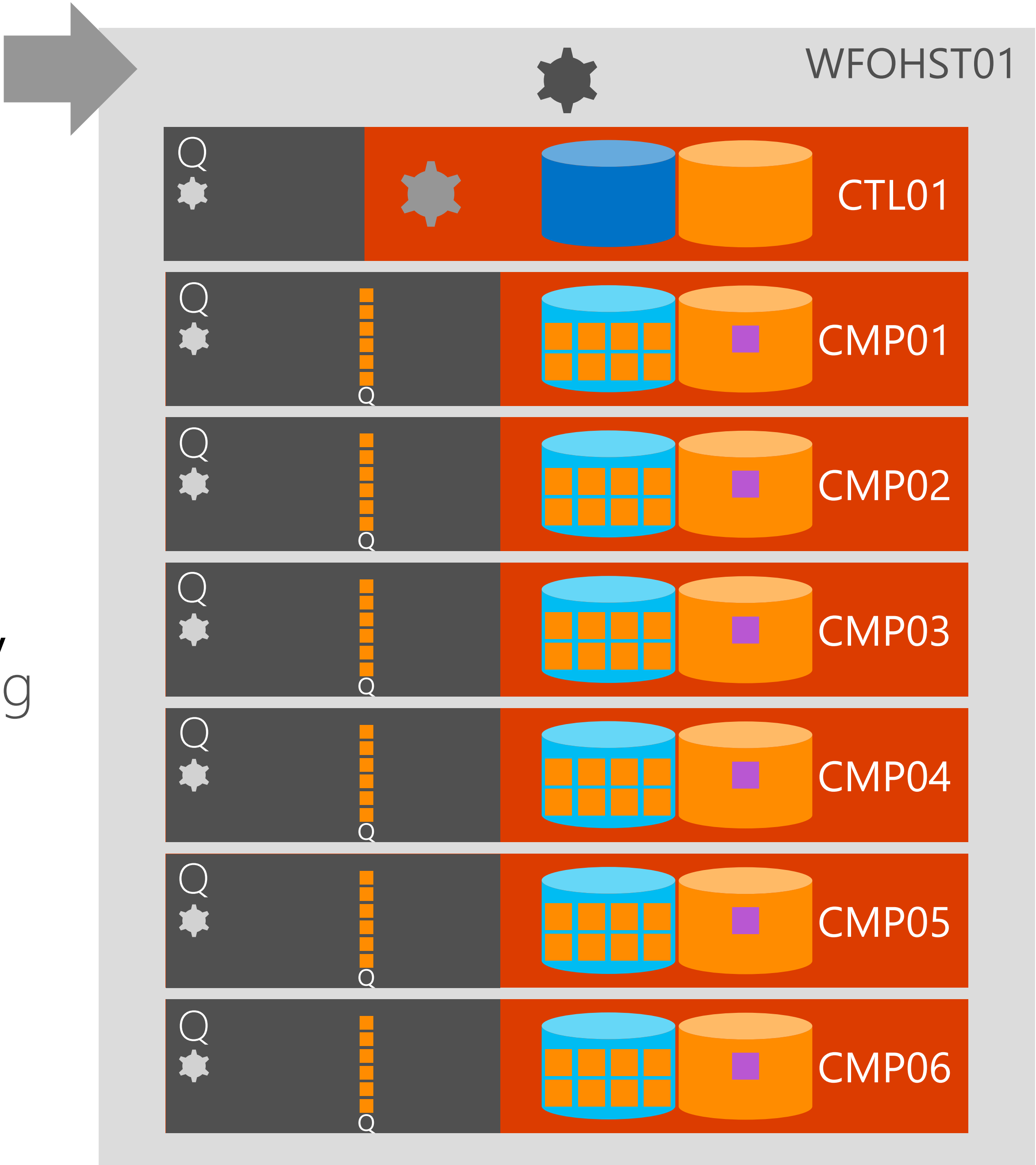
```
SELECT    c.FirstName + ' ' + c.LastName
FROM      dbo.FactInternetSales fis
LEFT JOIN dbo.DimCustomer c
ON        fis.CustomerKey=c.CustomerKey
WHERE     c.LastName = 'Smith'
OPTION (LABEL = 'Broadcast')
;
```

PDWEngine

- Creates a DSQL plan, by querying the shell database
- Generates Random ID
- Creates geometry aligned temp tables
- Instructs DMS to read data by executing a dsql query against the distributions

DMS

- Executes query from PDWEngine
- Query filters and pre-aggregates data
- Receives data in 32KB buffers
- Places buffers in a read queue for processing



Broadcast – Part 2

DMS Direct_Reader worker

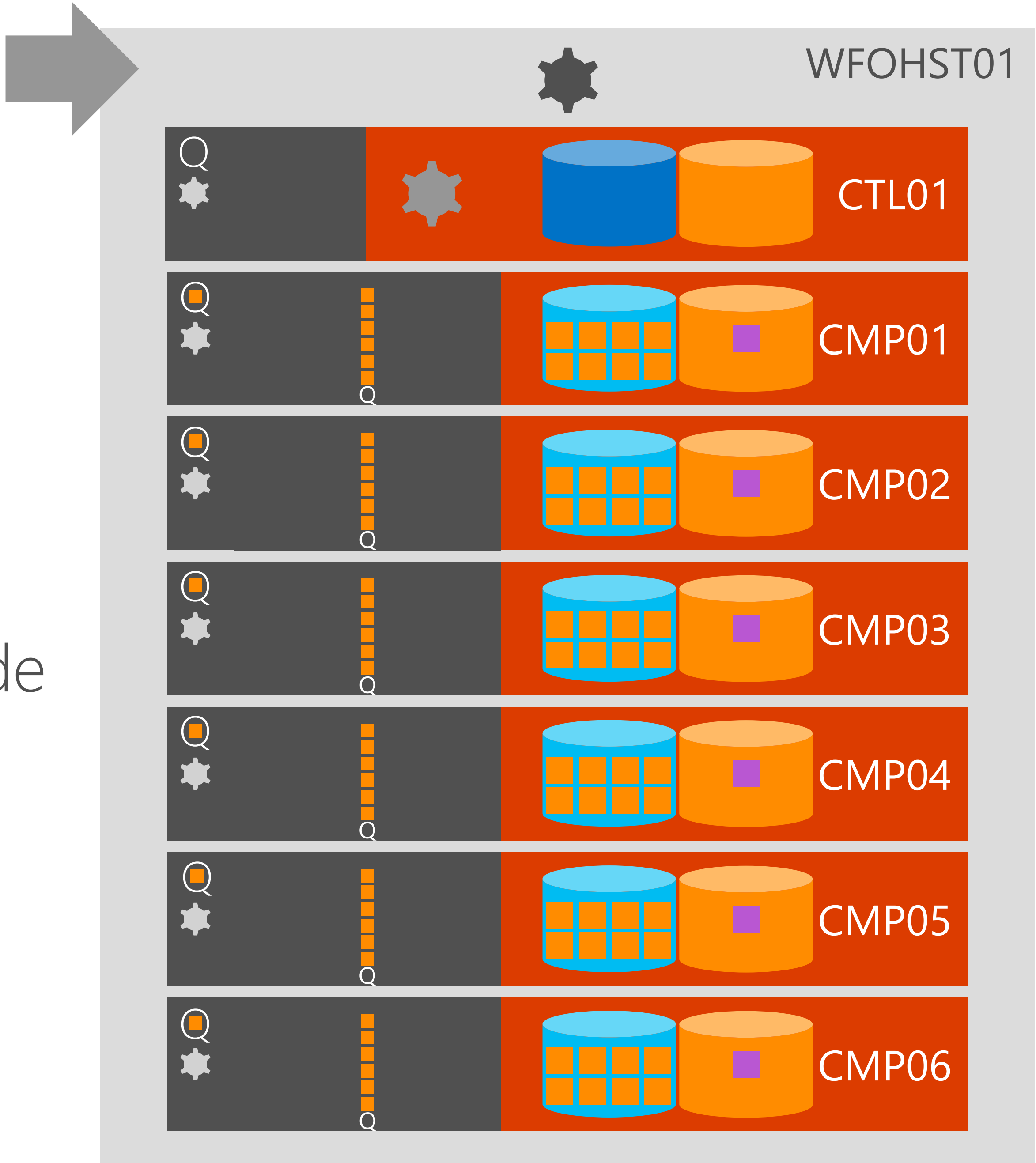
- Take buffer from queue
- Read buffer row by row
- Write row into new geometry aligned buffer

Sender

- Transmits full buffers to target node

Receiver

- Place buffer on geometry aligned write queue



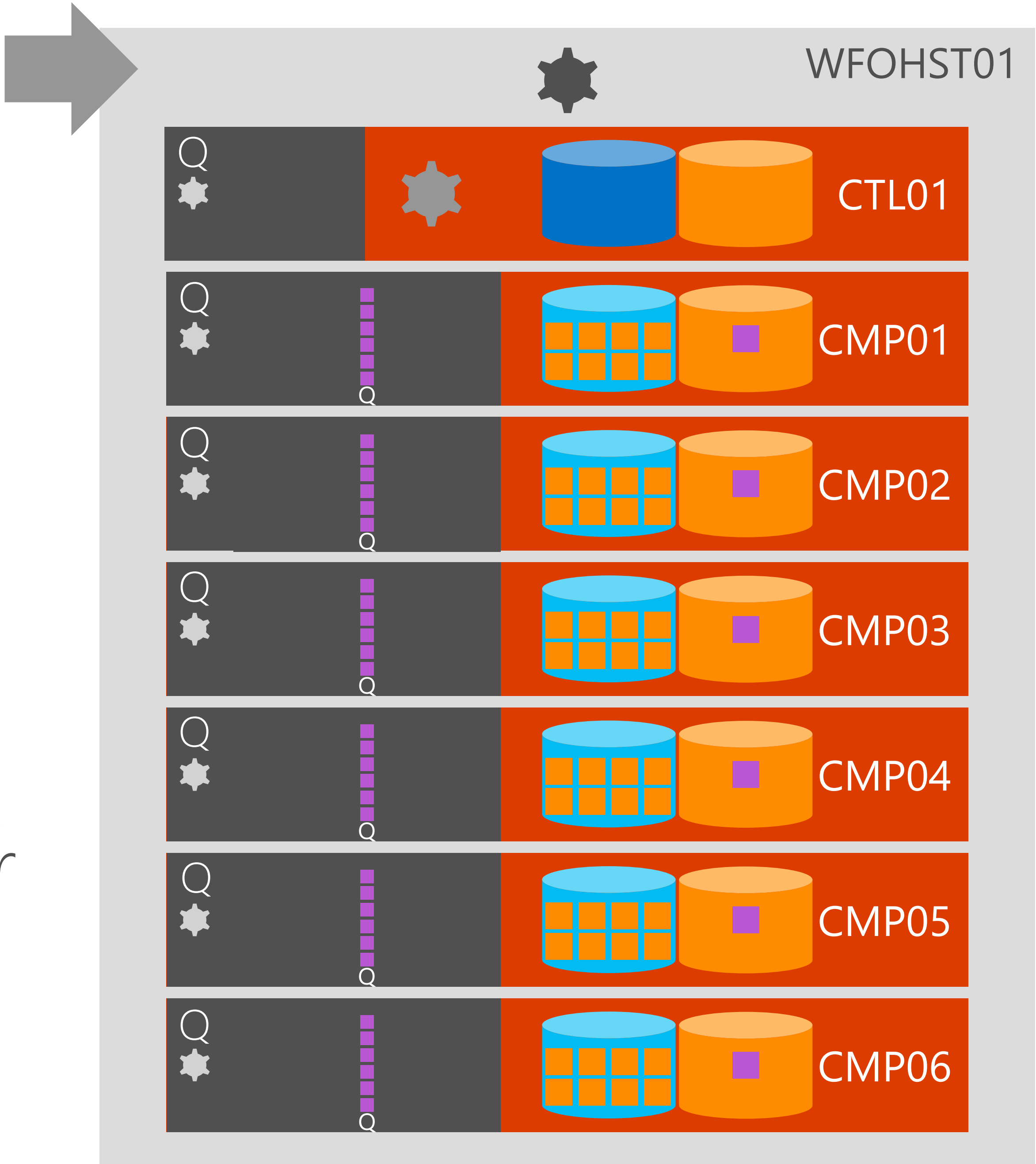
Broadcast – Part 3

DMS Writer Worker

- Bulk load buffers into geometry aligned tempdb "Q" tables

PDW Engine

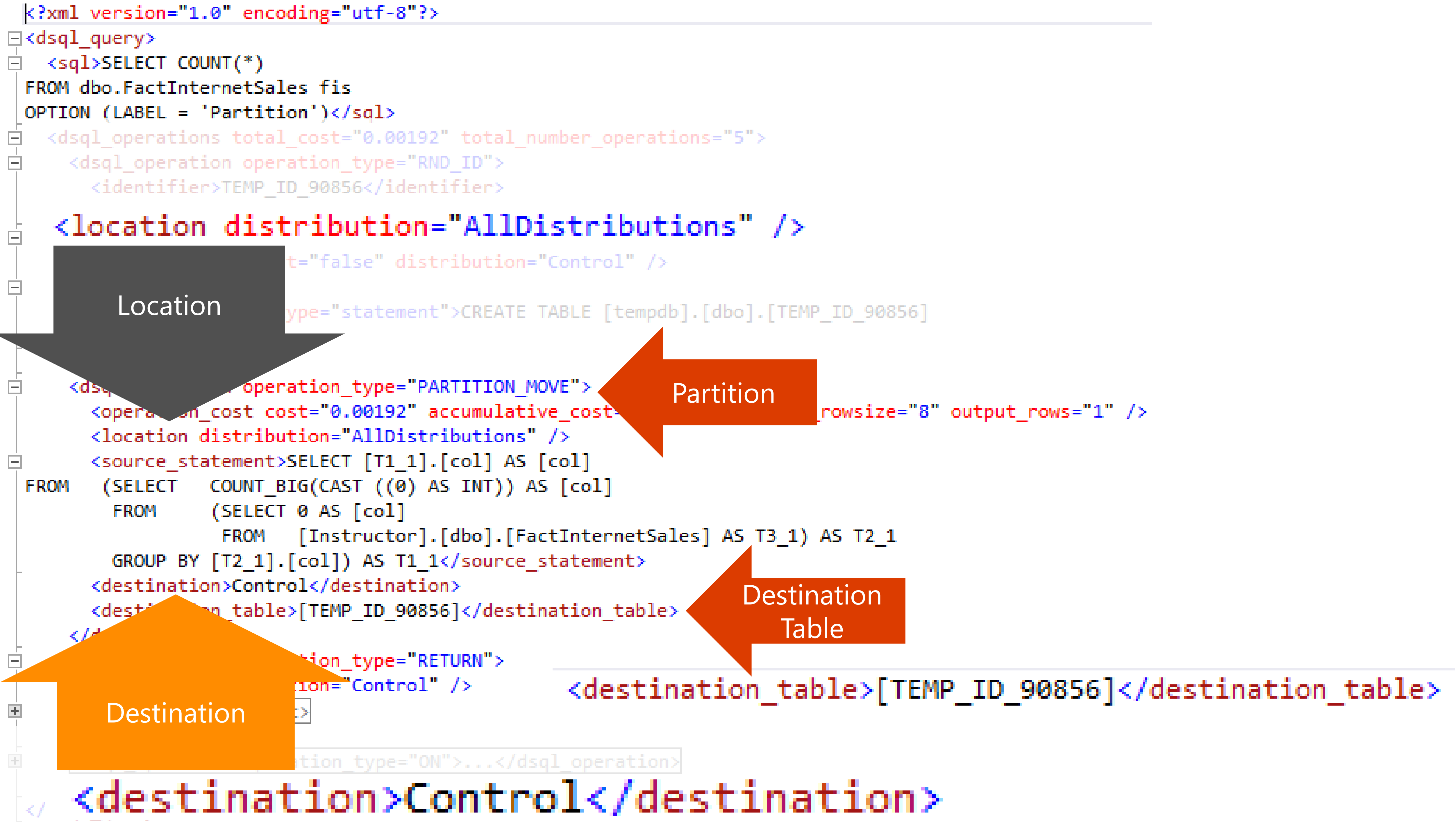
- Submits distribution compatible query against Q tables returning data to user
- Drops Q Tables



Partition Move Scenario

Understanding the Partition Move

- Centralise data to the control node
- Used for final aggregation
- Also used for temporary storage of an aggregation prior to another move operation
- No hashing of data required for Partition Move
- Worker type is PARALLEL_COPY_READER
- Can be an expensive operation
- Control node can become the bottleneck



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>SELECT COUNT(*)
FROM dbo.FactInternetSales fis
OPTION (LABEL = 'Partition')</sql>
  <dsql_operations total_cost="0.00192" total_number_operations="5">
    <dsql_operation operation_type="RND_ID">
      <identifier>TEMP_ID_90856</identifier>

                          t="false" distribution="Control" />

                          ype="statement">CREATE TABLE [tempdb].[dbo].[TEMP_ID_90856]


    <ds                   operation_type="PARTITION_MOVE">
      <opera   n_cost cost="0.00192" accumulative_cost                    rowsize="8" output_rows="1" />
      <location distribution="AllDistributions" />
      <source_statement>SELECT [T1_1].[col] AS [col]
FROM   (SELECT   COUNT_BIG(CAST ((0) AS INT)) AS [col]
        FROM     (SELECT 0 AS [col]
                  FROM   [Instructor].[dbo].[FactInternetSales] AS T3_1) AS T2_1
        GROUP BY [T2_1].[col]) AS T1_1</source_statement>
      <destination>Control</destination>
      <dest     n_table>[TEMP_ID_90856]</destination_table>
    </d
                          tion_type="RETURN">
                          ion="Control" />
                           >

                          tion_type="ON">...</dsql_operation>
  </
```

<location distribution="AllDistributions" />

Location

Partition

Destination Table

<destination_table>[TEMP_ID_90856]</destination_table>

Destination

<destination>Control</destination>

Partition – Part 1

User submits query

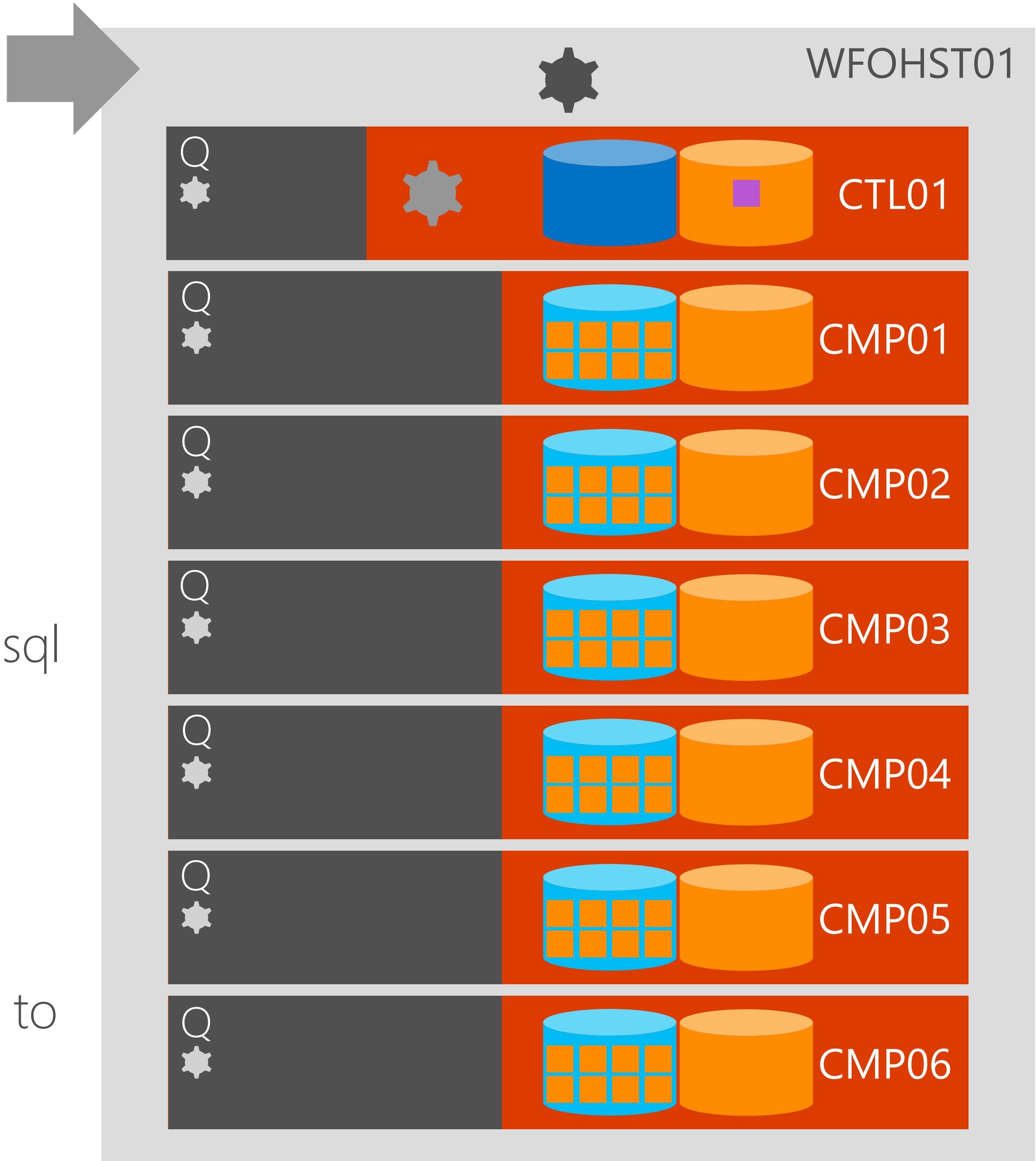
PDWEngine

- Creates a DSQL plan, by querying the shell database
- Generates Random ID
- Creates temp table on Control Node
- Instructs DMS to read data by executing a dsql query against the distributions

```
SELECT COUNT(*)
FROM dbo.FactInternetSales fis
OPTION (LABEL = 'Partition')
```

DMS

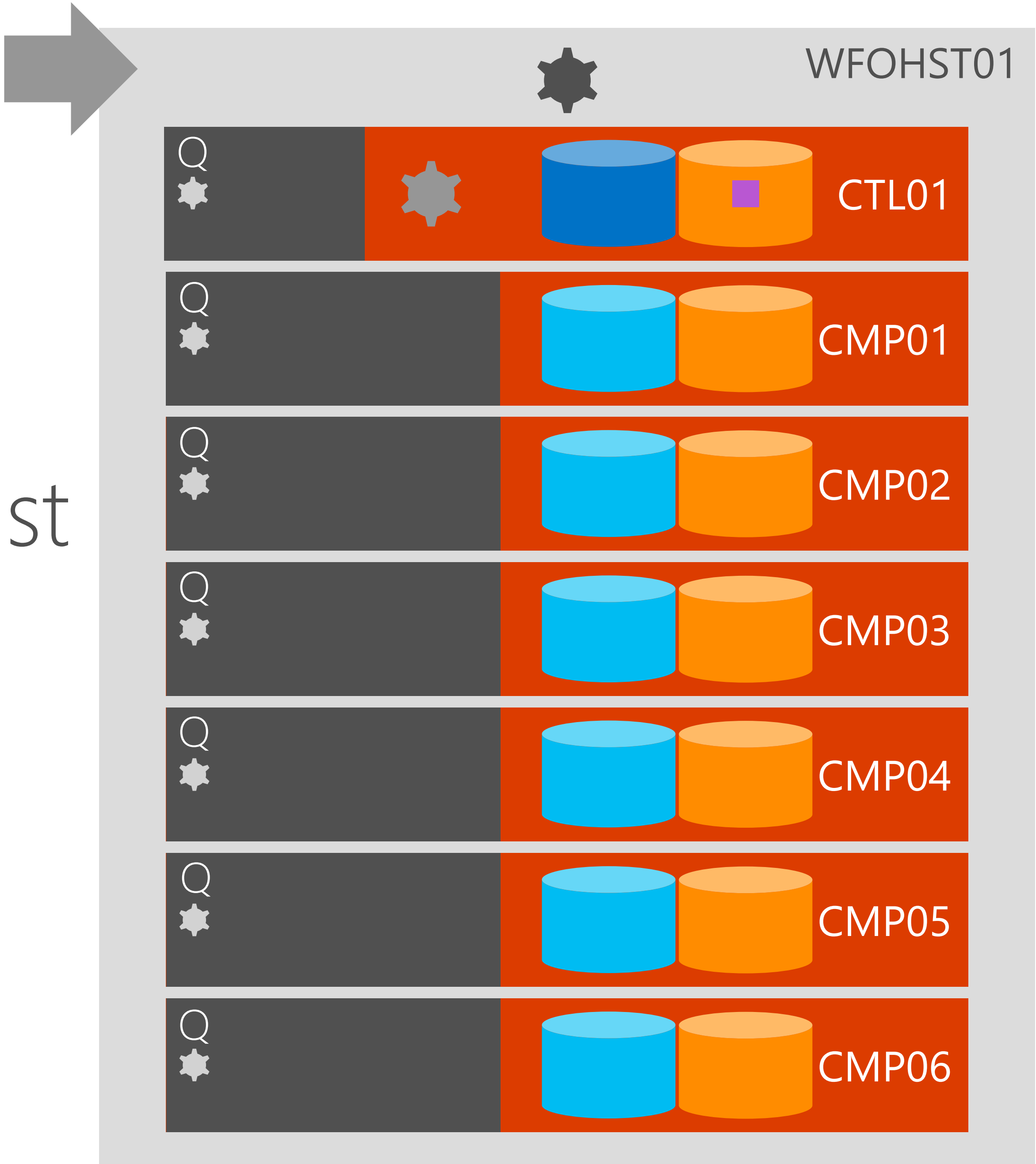
- Executes query from PDWEngine
- Query filters and pre-aggregates data
- Process data using Parallel_Copy_Reader worker
- Bypass data channel writing the row directly to Tempdb on control node



Partition – Part 2

PDWEngine

- Submits distribution compatible query against the Q table returning data to user
- Drops Q table

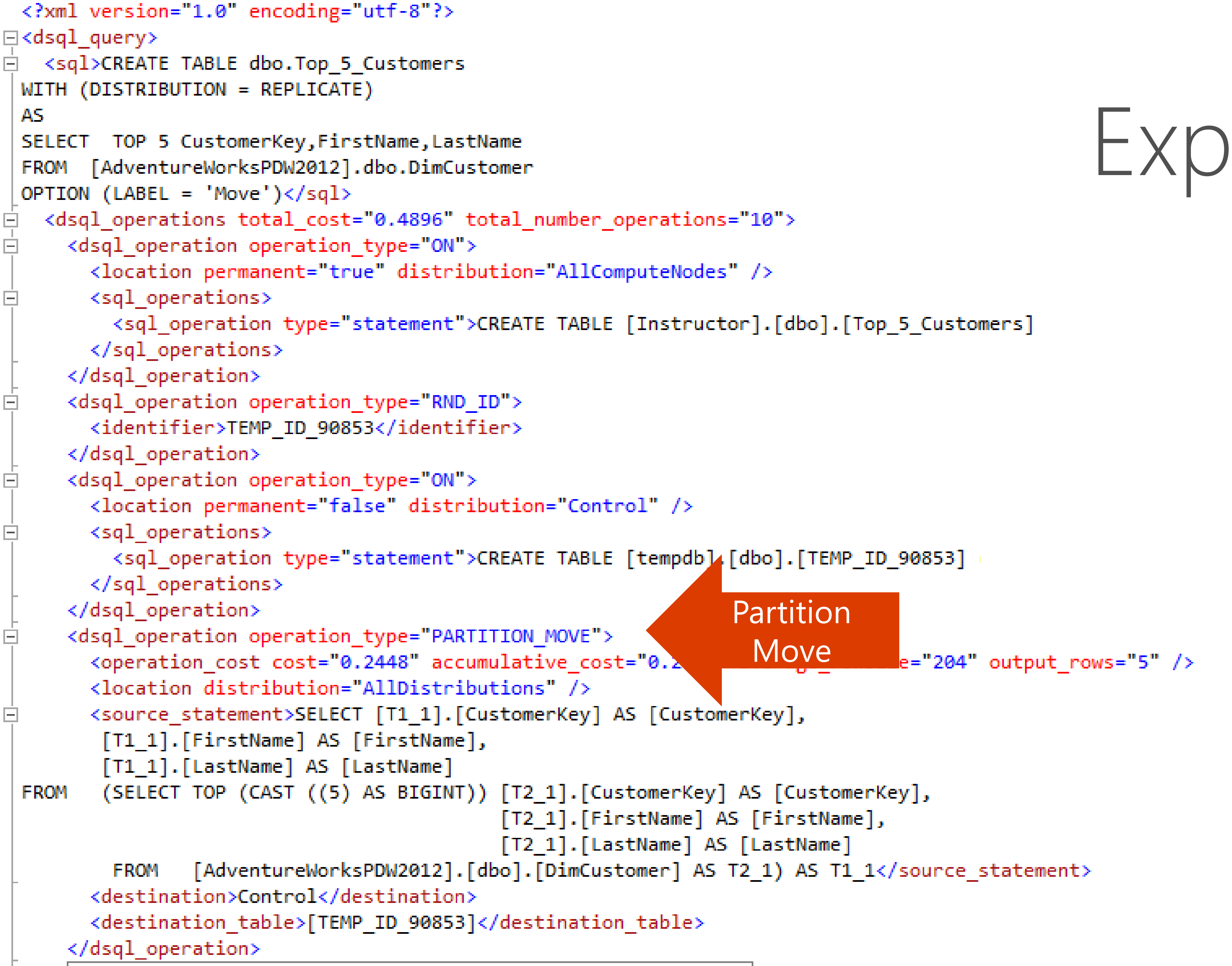


Master Move Scenario

Understanding The Master Move

- No user data is allowed to persist on the control node
- A mechanism is required to push the data back to the compute nodes
- A move operation is <u>typically</u> preceded by a partition move

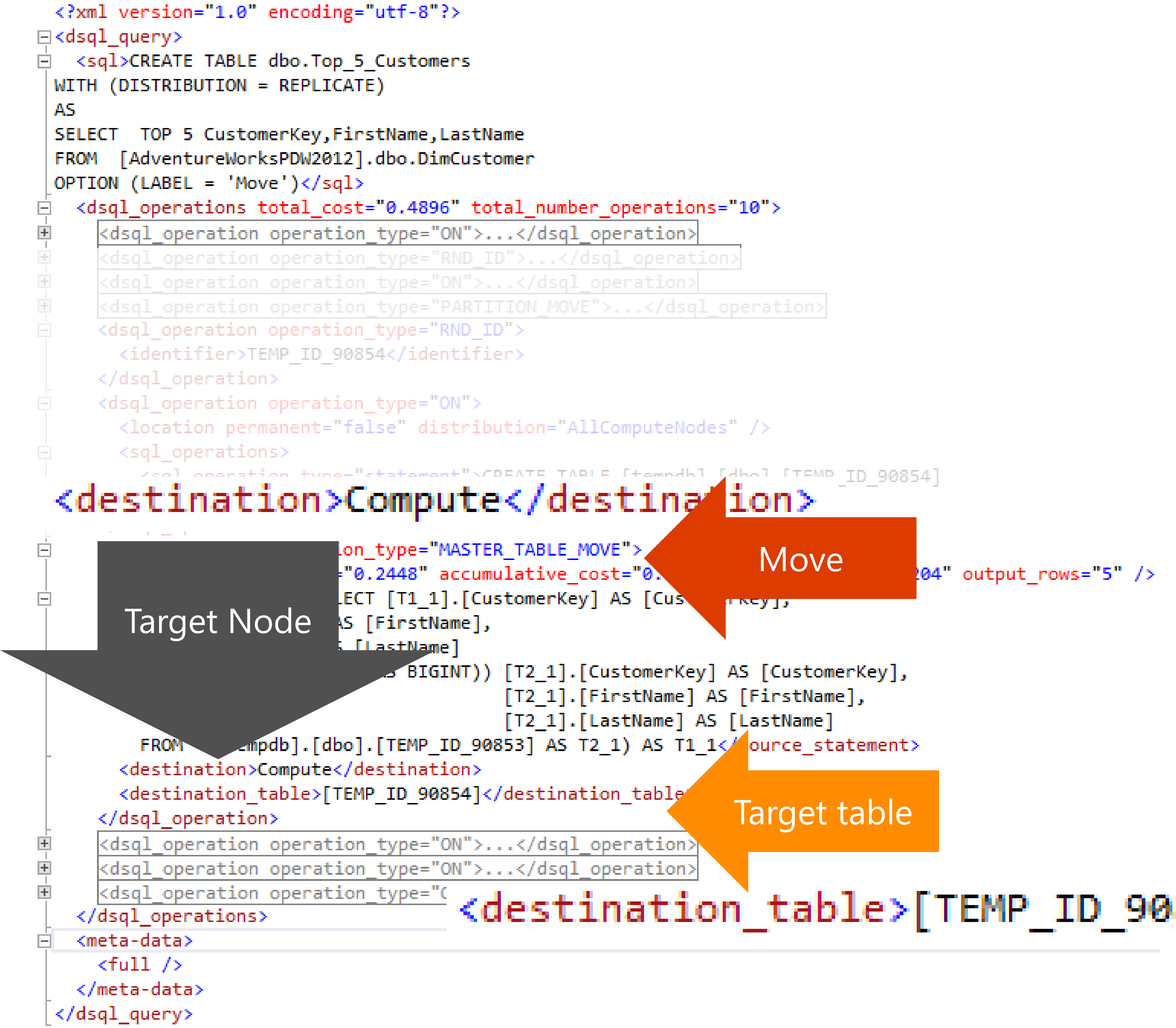
Explain Plan



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>CREATE TABLE dbo.Top_5_Customers
WITH (DISTRIBUTION = REPLICATE)
AS
SELECT  TOP 5 CustomerKey,FirstName,LastName
FROM  [AdventureWorksPDW2012].dbo.DimCustomer
OPTION (LABEL = 'Move')</sql>
  <dsql_operations total_cost="0.4896" total_number_operations="10">
    <dsql_operation operation_type="ON">
      <location permanent="true" distribution="AllComputeNodes" />
      <sql_operations>
        <sql_operation type="statement">CREATE TABLE [Instructor].[dbo].[Top_5_Customers]
      </sql_operations>
    </dsql_operation>
    <dsql_operation operation_type="RND_ID">
      <identifier>TEMP_ID_90853</identifier>
    </dsql_operation>
    <dsql_operation operation_type="ON">
      <location permanent="false" distribution="Control" />
      <sql_operations>
        <sql_operation type="statement">CREATE TABLE [tempdb].[dbo].[TEMP_ID_90853]
      </sql_operations>
    </dsql_operation>
    <dsql_operation operation_type="PARTITION_MOVE">
      <operation_cost cost="0.2448" accumulative_cost="0.2                 e="204" output_rows="5" />
      <location distribution="AllDistributions" />
      <source_statement>SELECT [T1_1].[CustomerKey] AS [CustomerKey],
       [T1_1].[FirstName] AS [FirstName],
       [T1_1].[LastName] AS [LastName]
FROM   (SELECT TOP (CAST ((5) AS BIGINT)) [T2_1].[CustomerKey] AS [CustomerKey],
                                          [T2_1].[FirstName] AS [FirstName],
                                          [T2_1].[LastName] AS [LastName]
        FROM   [AdventureWorksPDW2012].[dbo].[DimCustomer] AS T2_1) AS T1_1</source_statement>
      <destination>Control</destination>
      <destination_table>[TEMP_ID_90853]</destination_table>
    </dsql_operation>
```

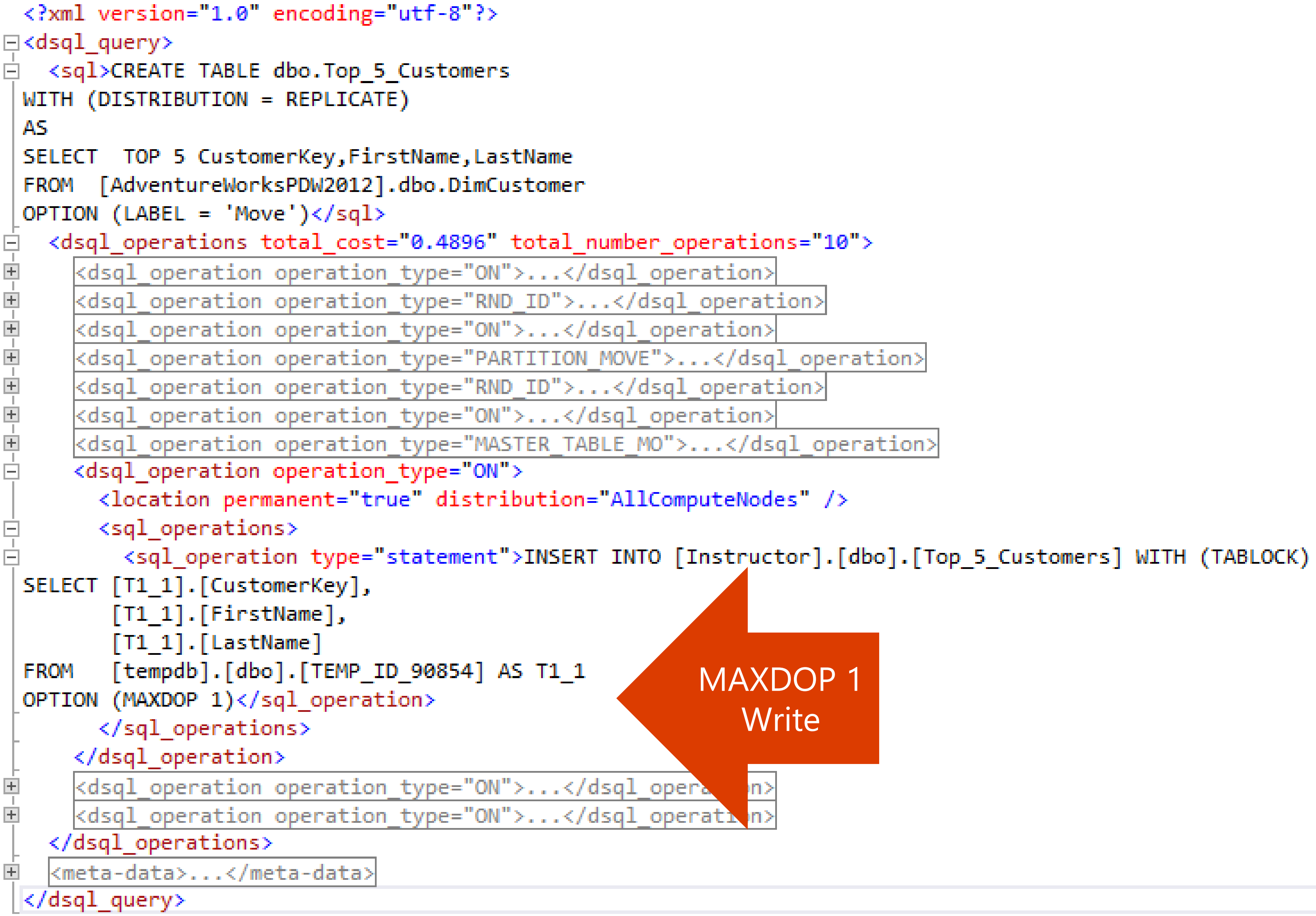
Partition Move

Explain Plan



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>CREATE TABLE dbo.Top_5_Customers
WITH (DISTRIBUTION = REPLICATE)
AS
SELECT  TOP 5 CustomerKey,FirstName,LastName
FROM  [AdventureWorksPDW2012].dbo.DimCustomer
OPTION (LABEL = 'Move')</sql>
  <dsql_operations total_cost="0.4896" total_number_operations="10">
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="RND_ID">...</dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="PARTITION_MOVE">...</dsql_operation>
    <dsql_operation operation_type="RND_ID">
      <identifier>TEMP_ID_90854</identifier>
    </dsql_operation>
    <dsql_operation operation_type="ON">
      <location permanent="false" distribution="AllComputeNodes" />
      <sql_operations>
        <sql_operation type="statement">CREATE TABLE [tempdb].[dbo].[TEMP_ID_90854]
<destination>Compute</destination>
                 on_type="MASTER_TABLE_MOVE">
                 ="0.2448" accumulative_cost="0.                 204" output_rows="5" />
                 ECT [T1_1].[CustomerKey] AS [Cus        Key],
                 AS [FirstName],
                   [LastName]
                   BIGINT)) [T2_1].[CustomerKey] AS [CustomerKey],
                            [T2_1].[FirstName] AS [FirstName],
                            [T2_1].[LastName] AS [LastName]
      FROM      mpdb].[dbo].[TEMP_ID_90853] AS T2_1) AS T1_1</ ource_statement>
      <destination>Compute</destination>
      <destination_table>[TEMP_ID_90854]</destination_table
    </dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="(
  </dsql_operations>
  <meta-data>
    <full />
  </meta-data>
</dsql_query>
```

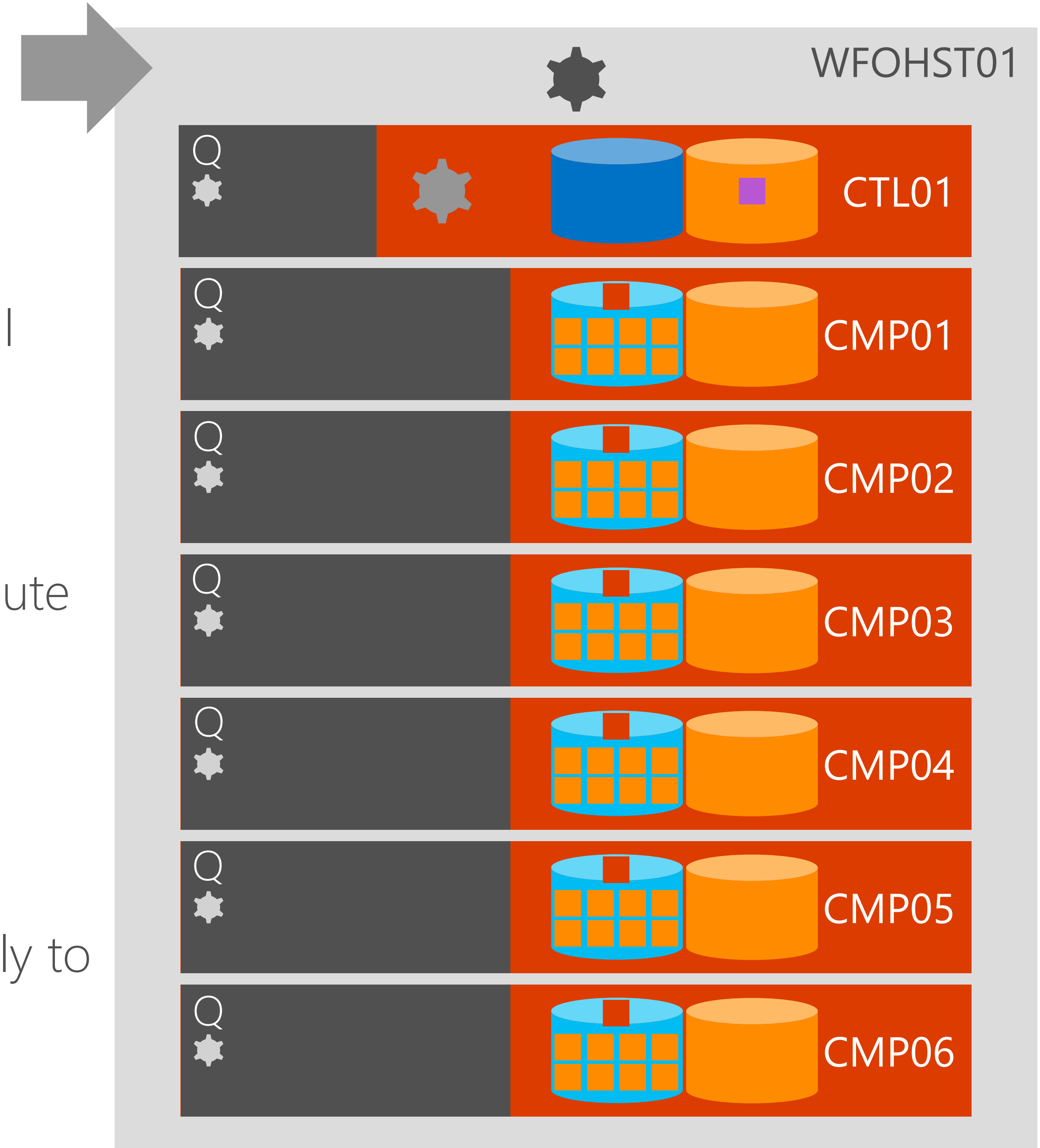
```
<destination_table>[TEMP_ID_90854]</destination_table>
```



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>CREATE TABLE dbo.Top_5_Customers
WITH (DISTRIBUTION = REPLICATE)
AS
SELECT  TOP 5 CustomerKey,FirstName,LastName
FROM  [AdventureWorksPDW2012].dbo.DimCustomer
OPTION (LABEL = 'Move')</sql>
  <dsql_operations total_cost="0.4896" total_number_operations="10">
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="RND_ID">...</dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="PARTITION_MOVE">...</dsql_operation>
    <dsql_operation operation_type="RND_ID">...</dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="MASTER_TABLE_MO">...</dsql_operation>
    <dsql_operation operation_type="ON">
      <location permanent="true" distribution="AllComputeNodes" />
      <sql_operations>
        <sql_operation type="statement">INSERT INTO [Instructor].[dbo].[Top_5_Customers] WITH (TABLOCK)
SELECT [T1_1].[CustomerKey],
       [T1_1].[FirstName],
       [T1_1].[LastName]
FROM   [tempdb].[dbo].[TEMP_ID_90854] AS T1_1
OPTION (MAXDOP 1)</sql_operation>
      </sql_operations>
    </dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
  </dsql_operations>
  <meta-data>...</meta-data>
</dsql_query>
```

MAXDOP 1
Write

Move – Part 1



User submits query

```sql
CREATE TABLE dbo.Top_5_Customers
WITH (DISTRIBUTION = REPLICATE)
AS
SELECT TOP 5
CustomerKey,FirstName,LastName
FROM dbo.DimCustomer
OPTION (LABEL = 'Move')
;
```

PDWEngine

- Creates a DSQL plan, by querying the shell database
- Creates target table in user database
- Generates Random ID
- Creates temp table on Control Node
- Instructs DMS to read data from the compute nodes

DMS

- Executes query from PDWEngine
- Query filters and pre-aggregates data
- Process data using Parallel_Copy_Reader worker
- Bypass data channel writing the row directly to Tempdb on control node

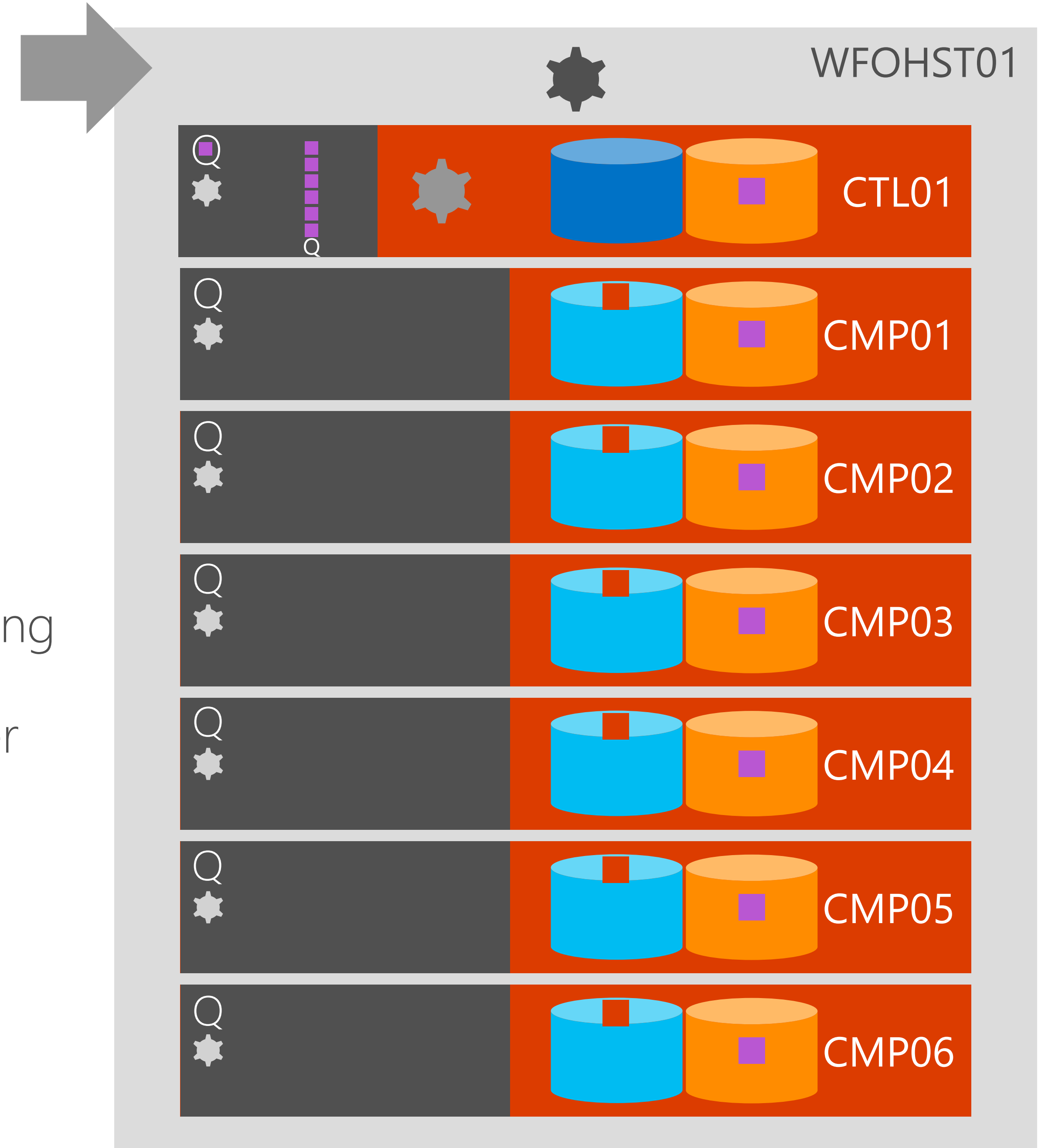
Move – Part 2

PDWEngine

- Generates Random ID
- Instruct DMS to read data held on control node

DMS

- Executes query from PDWEngine
- Receives data in 32KB buffers
- Places buffers in a read queue for processing
- Reads buffers using Direct_Reader worker
- Writes row to new geometry aligned buffer
- Places buffer on send queue



Move – Part 3

Sender

- Moves buffers to compute nodes via DMS data channel

Receiver

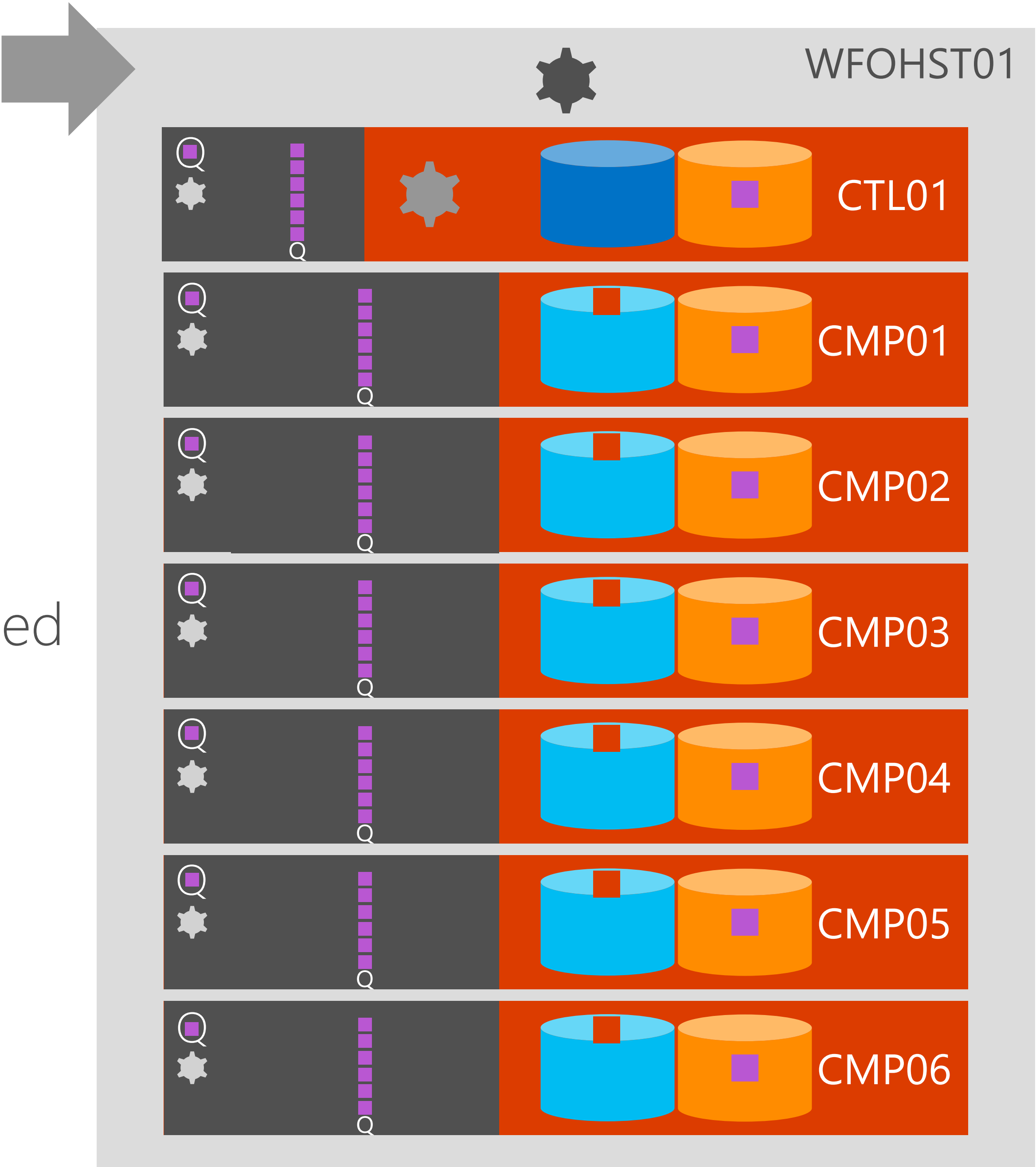
- Receives buffers and queues them

Writer worker

- Bulk loads buffers into geometry aligned tempdb "Q" tables

PDWEngine

- Submits MAXDOP(1) write queries to populate target table in user database
- Drops Q Tables

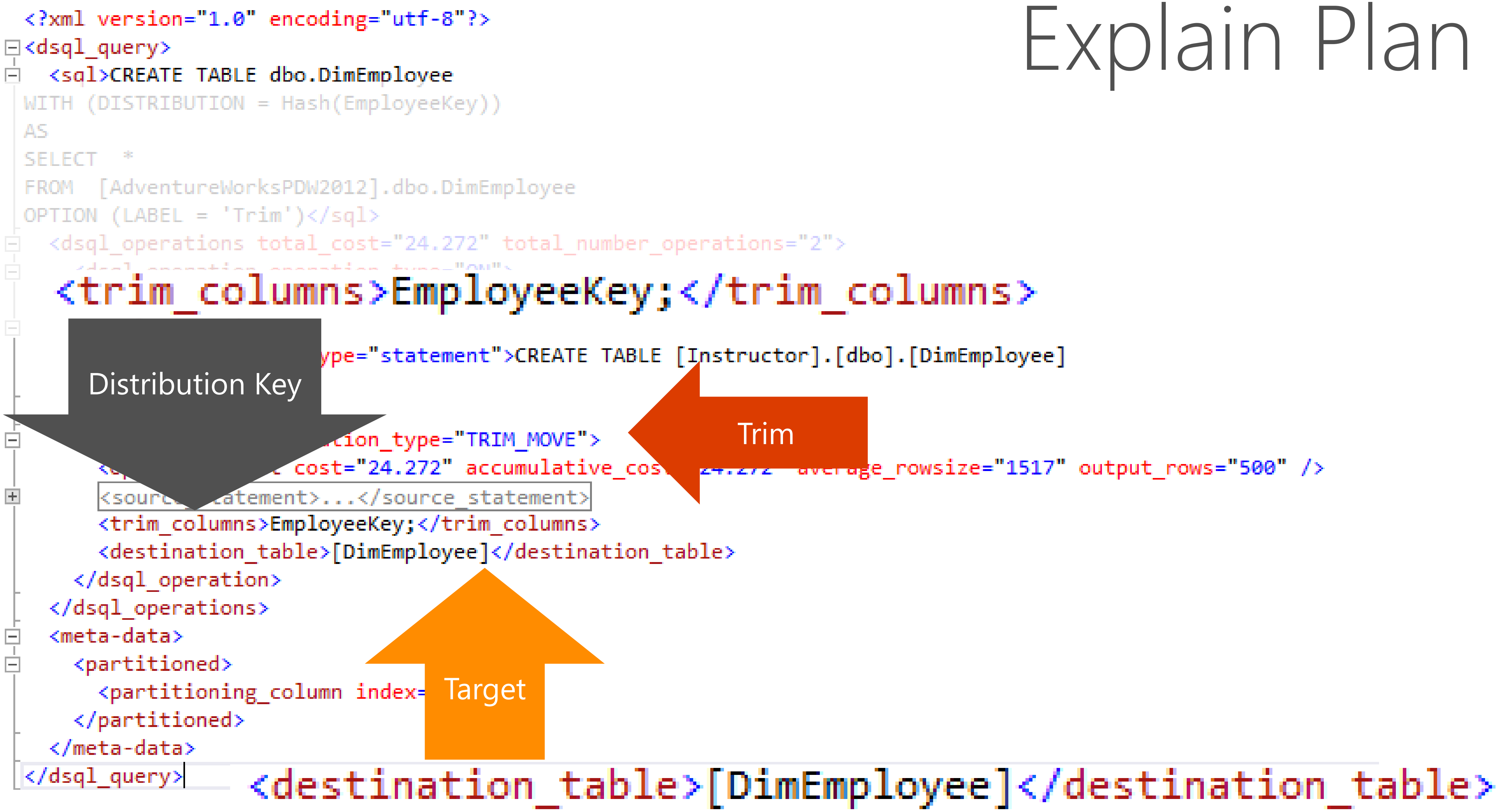


Trim Move Scenario

Introducing Trim Move

- Changing table geometry
- From a replicated table to distributed table
- All data exists on the compute node already
- No data leaves the compute node

Explain Plan



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>CREATE TABLE dbo.DimEmployee
WITH (DISTRIBUTION = Hash(EmployeeKey))
AS
SELECT  *
FROM  [AdventureWorksPDW2012].dbo.DimEmployee
OPTION (LABEL = 'Trim')</sql>
  <dsql_operations total_cost="24.272" total_number_operations="2">

              ype="statement">CREATE TABLE [Instructor].[dbo].[DimEmployee]

              tion_type="TRIM_MOVE">
      <               cost="24.272" accumulative_cos                average_rowsize="1517" output_rows="500" />
      <sourc    tatement>...</source_statement>
      <trim_columns>EmployeeKey;</trim_columns>
      <destination_table>[DimEmployee]</destination_table>
    </dsql_operation>
  </dsql_operations>
  <meta-data>
    <partitioned>
      <partitioning_column index=
    </partitioned>
  </meta-data>
</dsql_query>
```

<trim_columns>EmployeeKey;</trim_columns>

<destination_table>[DimEmployee]</destination_table>

Trim – Part 1

User submits query

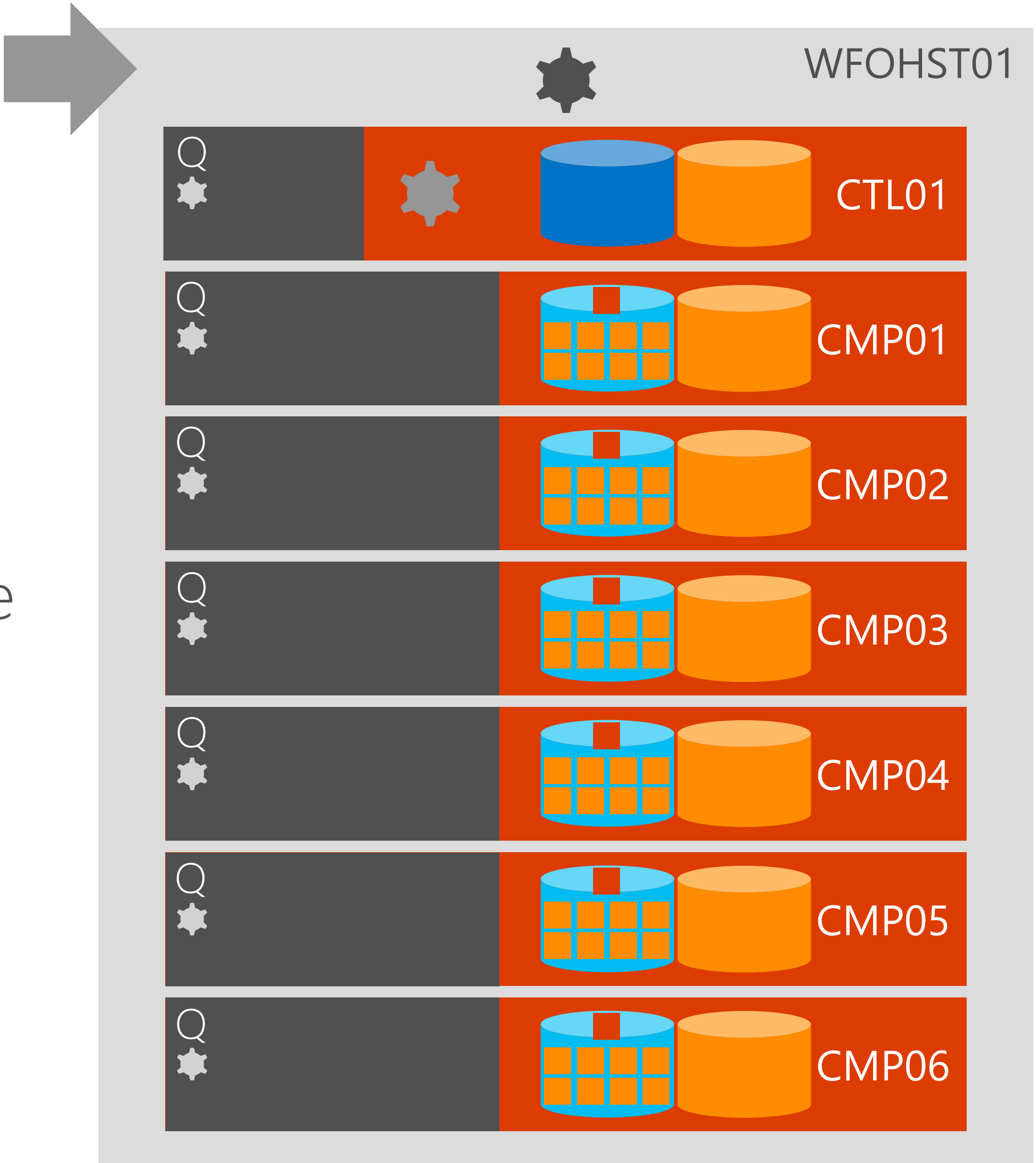
```
CREATE TABLE dbo.DimEmployee
WITH (DISTRIBUTION = Hash(EmployeeKey))
AS
SELECT *
FROM
[AdventureWorksPDW2012].dbo.DimEmployee
OPTION (LABEL = 'Trim')
;
```

PDWEngine

- Creates a DSQL plan, by querying the shell database
- Creates a distributed table in user database
- Instructs DMS to read data from the replicated table on each node

DMS

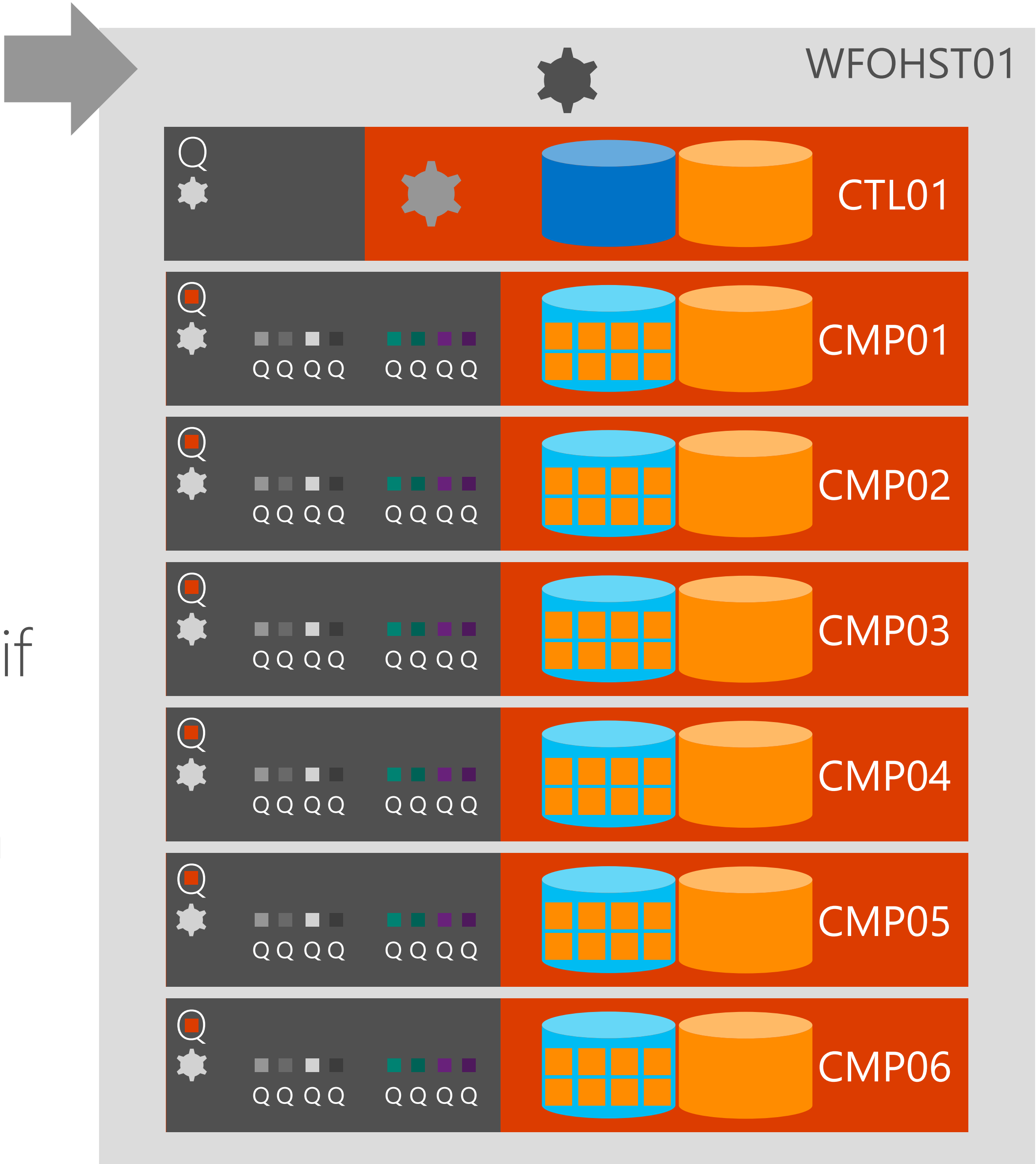
- Executes query from PDWEngine
- Receives data in 32KB buffers
- Places buffers a read queue for processing



Trim – Part 2

The DMS performs the Trim move

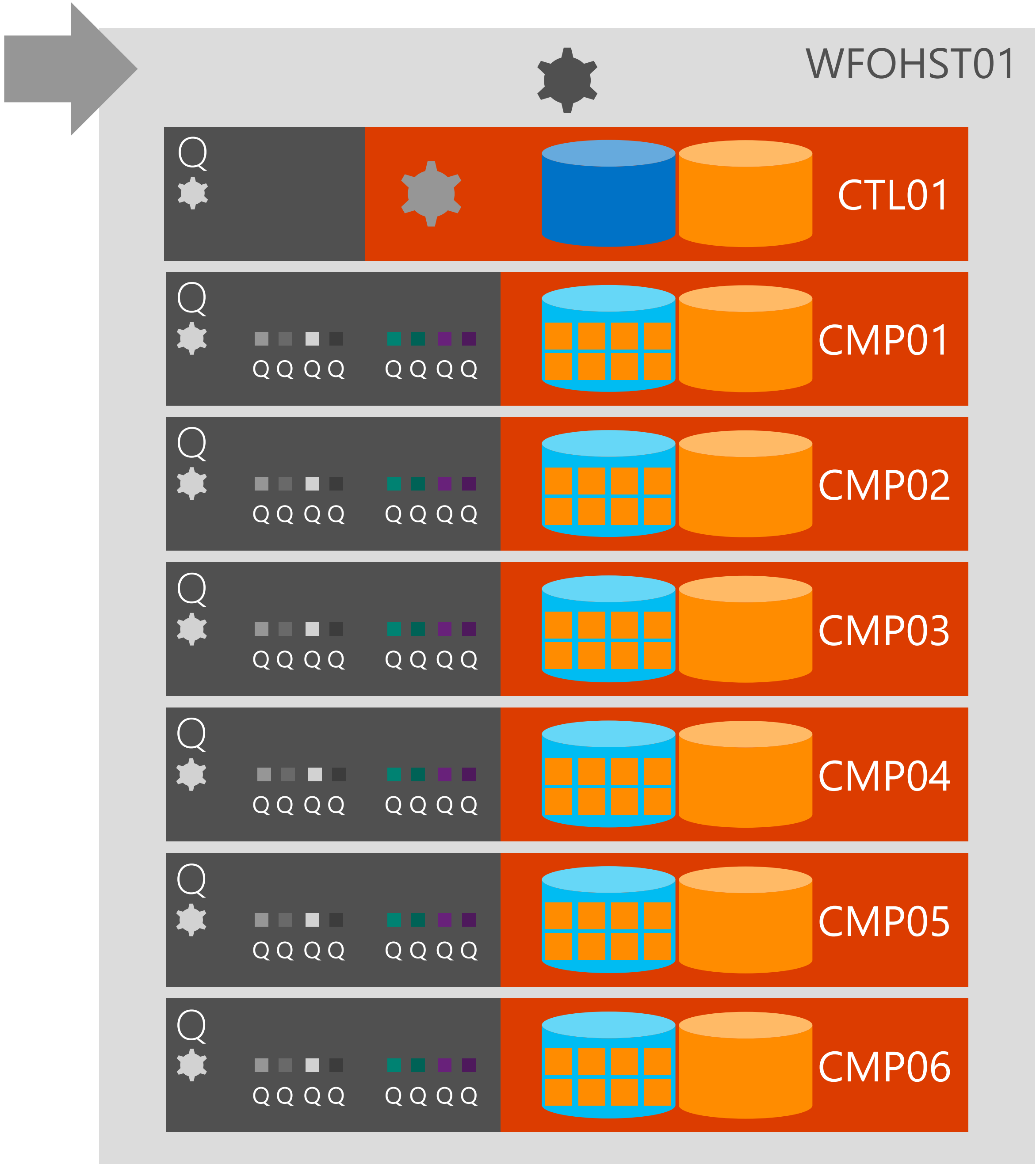
- Pull a buffer from queue
- Reads buffer row by row using Hash_Reader worker
- Hash on new distribution key
- Only writes rows to the new distribution aligned output buffer if the row belongs on this compute node
- When buffer is full place buffer on distribution aligned write queue ready for writing



Trim – Part 3

The DMS Writer Workers

- Pull buffers from the write queue
- Bulk loads the data directly into the target tables

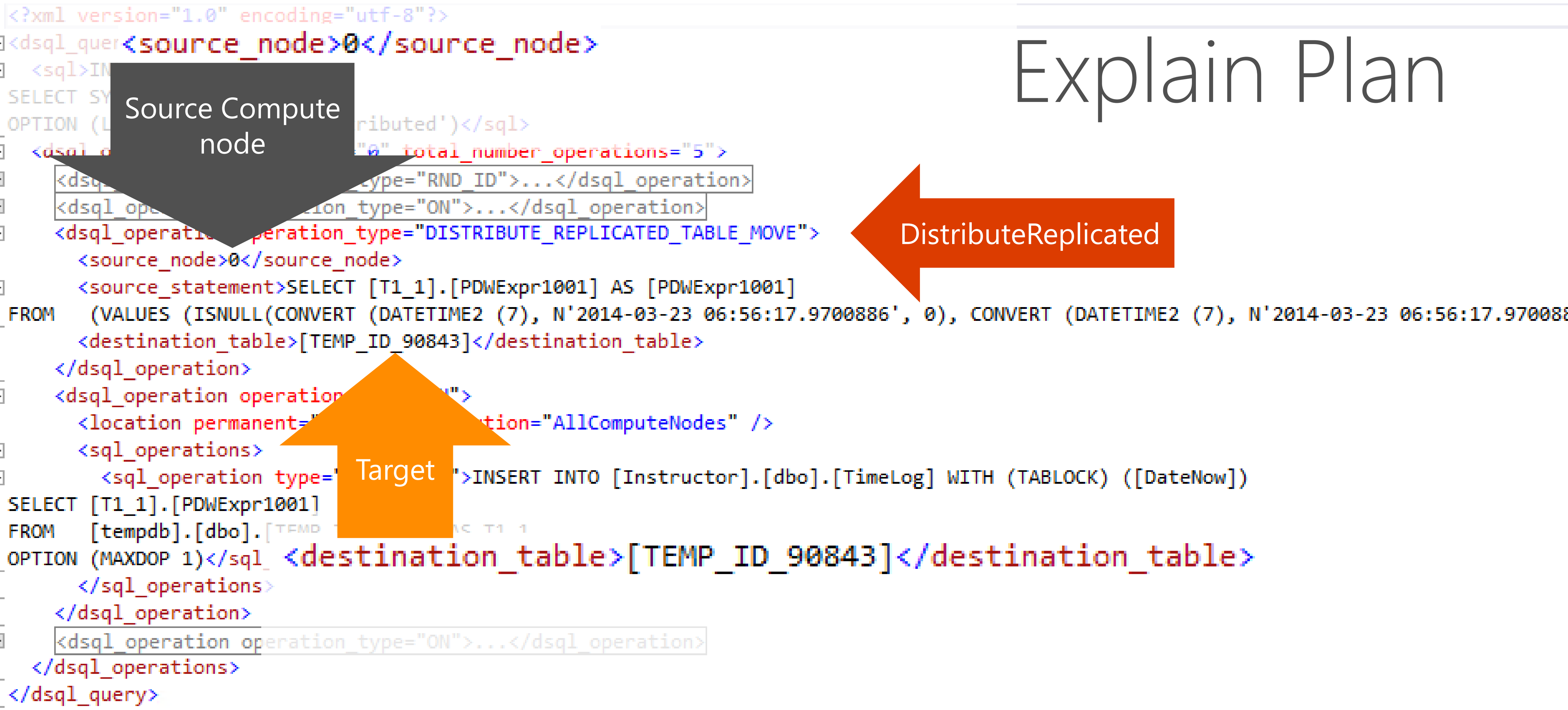


DistributeReplicatedTable Move Scenario

Introducing DistributeReplicatedTable Move

- Required to provide consistency across replicated tables
- Write is first persisted on one compute node in Tempdb
- Persisted data is then shared with other nodes by bulk inserting the values into their Tempdb
- DistributeReplicatedTable DMS movement does not insert data into the target
  - Performed by PDWEngine as a follow up step

Explain Plan



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_quer<source_node>0</source_node>
  <sql>IN
SELECT SY
OPTION (L                      ributed')</sql>
  <dsql_o                      "0" total_number_operations="5">
    <dsq                          type="RND_ID">...</dsql_operation>
    <dsql_ope              ion_type="ON">...</dsql_operation>
    <dsql_operati          peration_type="DISTRIBUTE_REPLICATED_TABLE_MOVE">
      <source_node>0</source_node>
      <source_statement>SELECT [T1_1].[PDWExpr1001] AS [PDWExpr1001]
FROM   (VALUES (ISNULL(CONVERT (DATETIME2 (7), N'2014-03-23 06:56:17.9700886', 0), CONVERT (DATETIME2 (7), N'2014-03-23 06:56:17.970088
      <destination_table>[TEMP_ID_90843]</destination_table>
    </dsql_operation>
    <dsql_operation operation              ">
      <location permanent="                 tion="AllComputeNodes" />
      <sql_operations>
        <sql_operation type="              ">INSERT INTO [Instructor].[dbo].[TimeLog] WITH (TABLOCK) ([DateNow])
SELECT [T1_1].[PDWExpr1001]
FROM   [tempdb].[dbo].[TEMP                AS T1_1
OPTION (MAXDOP 1)</sql <destination_table>[TEMP_ID_90843]</destination_table>
      </sql_operations>
    </dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
  </dsql_operations>
</dsql_query>
```

Source Compute node

DistributeReplicated

Target

DistributeReplicated – Part 1

User submits write query

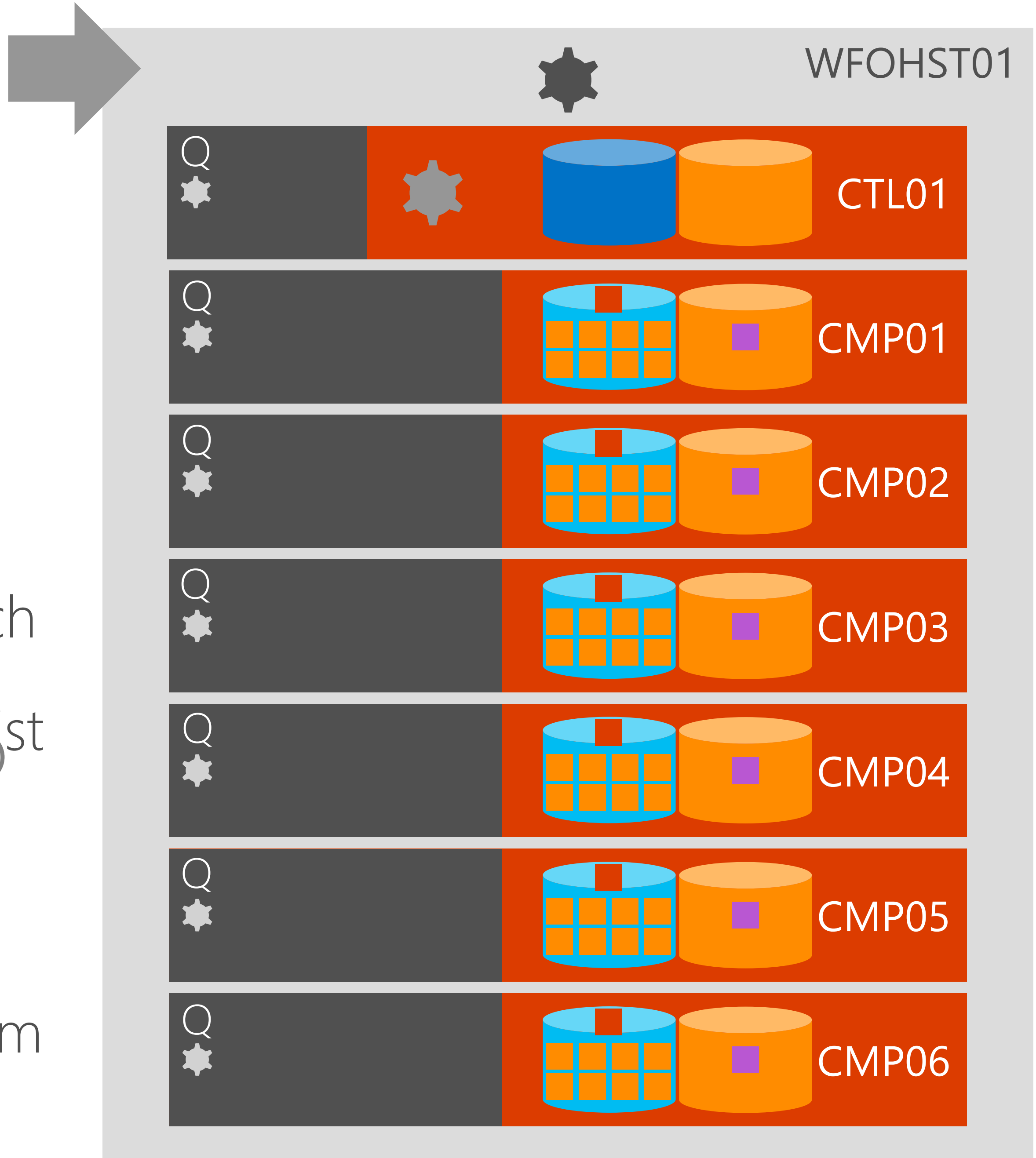
PDWEngine

- Creates a DSQL plan, by querying the shell database
- Generates a Random ID
- Creates a replicated temp table on each node
- Writes data into one node first to persist the values and act as the data source

DMS

- Executes query from PDWEngine
- Reads data using Hash_Reader worker
- Holds data in 32KB buffers placing them in a read queue for processing

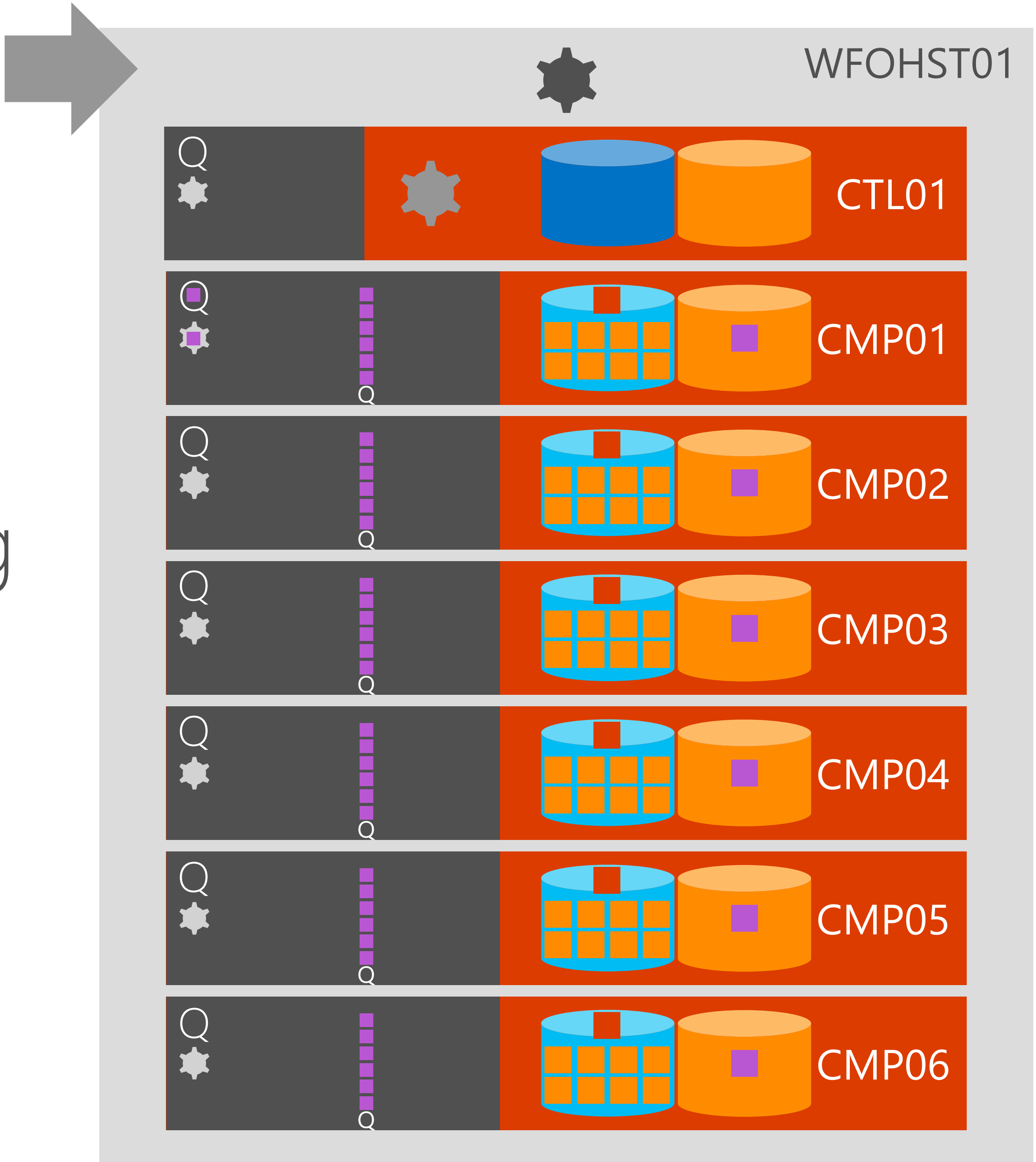
```
CREATE TABLE TimeLog
(DateNow datetime2)

INSERT INTO TimeLog
SELECT SYSDATETIME()
OPTION (LABEL = 'ReplicateDistributed')
;
```



DistributeReplicated – Part 2

The DMS performs the DistributeReplicated move

- Pull buffers from the read queue on the node acting as the source
- Distributes the data to all compute nodes placing it onto the read queue



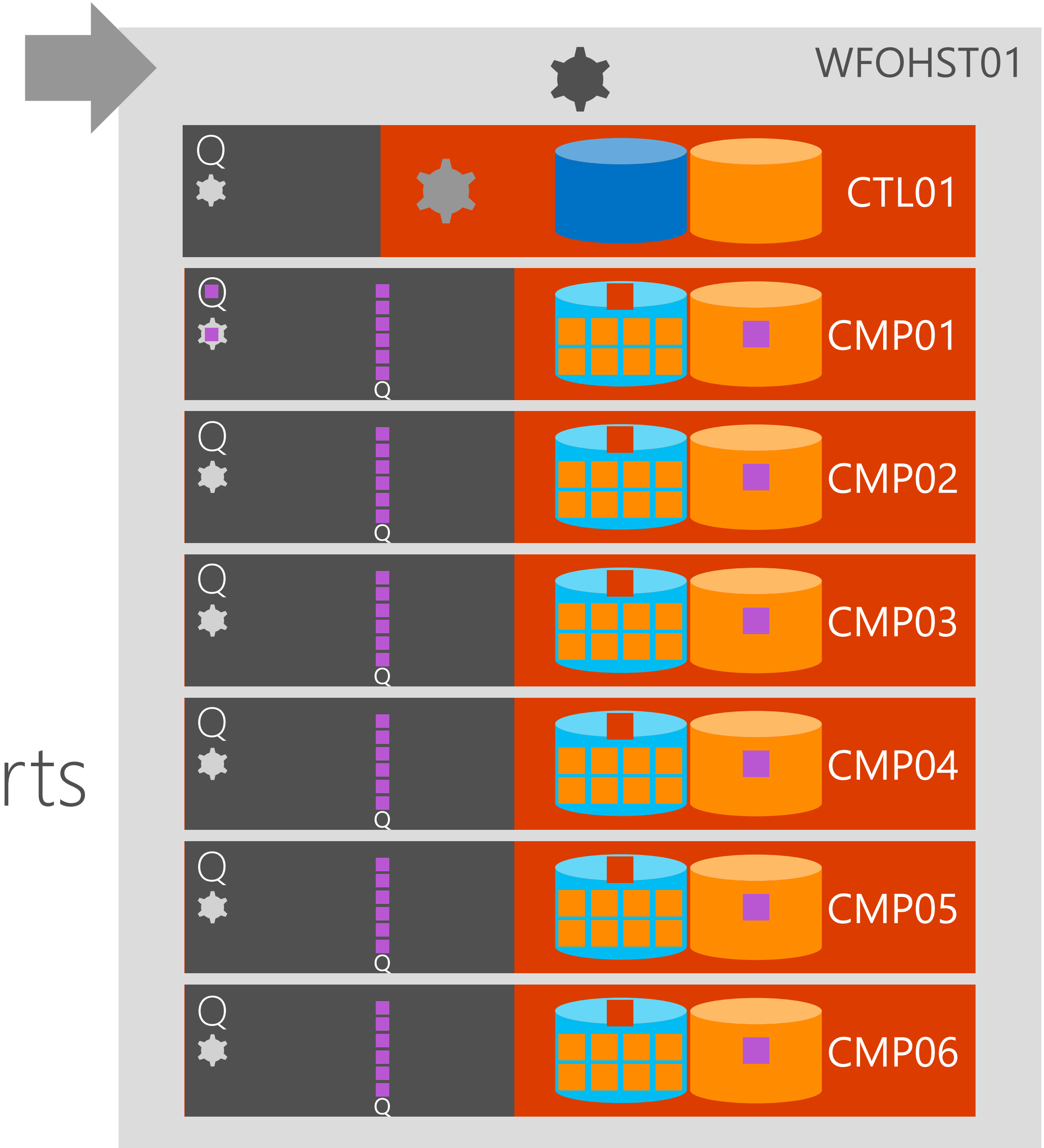
DistributeReplicated – Part 3

The DMS Writer Worker

- Pulls buffers from write queue
- Bulk load buffers into Tempdb "Q" table

PDWEngine

- Submits MAXDOP(1) inserts populating target table
- Drops tables in Tempdb



Optimising DMS

Buffer Density

- Buffers used by query processing max 32KB
- Whole rows must fit into each buffer
  - No overflow
- All columns exploded to max size in buffer
  - i.e. VARCHAR(4000) = 4000 bytes in the buffer
- Nullable columns add eight bytes to column

Wide variable length character columns really hurt buffer density

Rationalise data types where possible avoiding nullable columns

Large numbers of nullable columns add up quickly. Akin to death by a thousand cuts!

Calculating Density

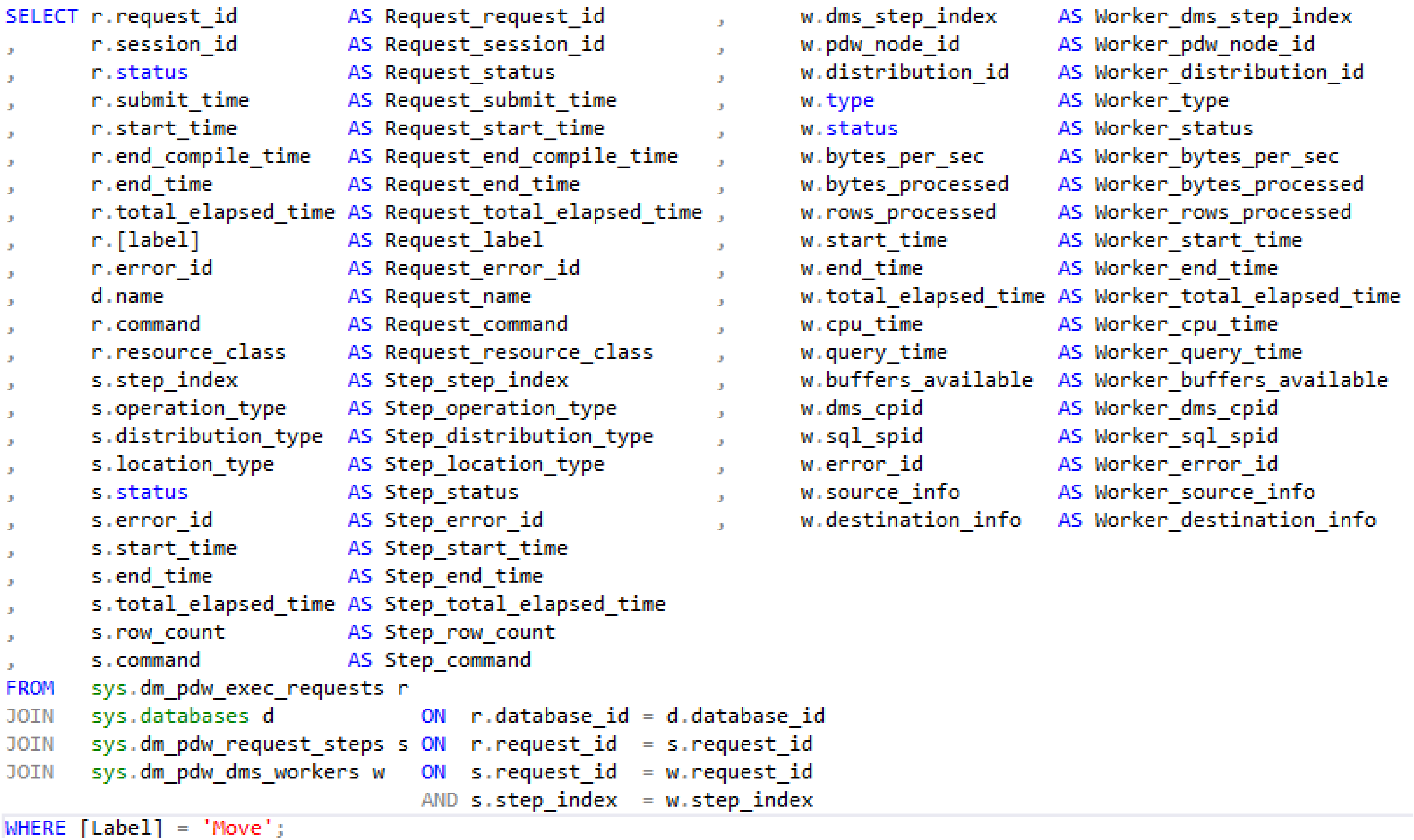
```
WITH T AS
(
SELECT t.name          AS [Table_Name]
,      c.name          AS [Column_Name]
,      c.is_nullable   AS [NullValue]
,      CASE WHEN ty.name IN ('char','varchar','nchar','nvarchar') THEN 1
            ELSE 0
       END AS [VariableLength]
,      CASE WHEN ty.name IN('char','varchar')     THEN c.max_length + 1
            WHEN ty.name IN('nchar','nvarchar')   THEN c.max_length + 2
            WHEN ty.name IN('binary','varbinary') THEN c.max_length
       ELSE ty.max_length
       END AS [DataLength]
FROM sys.tables t
JOIN sys.columns c on t.object_id = c.object_id
JOIN ( SELECT    name
        ,        system_type_id
        ,        user_type_id
        ,        CASE   WHEN name =  'time' THEN 12
                     WHEN name IN ('date','datetime','datetime2','datetimeoffset') THEN max_length * 2
                     WHEN name =  'smalldatetime' THEN 16
                     WHEN name =  'decimal' THEN 19
                     ELSE max_length
                END as max_length
FROM sys.types t
     ) ty  ON   c.system_type_id = ty.system_type_id
           AND  c.user_type_id   = ty.user_type_id
)
```

Calculating Density Part 2

```
SELECT [Table_Name]
,      CAST(32768. / SUM(CASE   WHEN NullValue = 1 OR VariableLength = 1 THEN 8
                                ELSE 0
                         END +[DataLength]) AS INT) AS RowsPerBuffer
,      SUM(CASE                 WHEN NullValue = 1 OR VariableLength = 1 THEN 8
                                ELSE 0
                         END + [DataLength])  AS RowSize
,      32768. % SUM(CASE        WHEN NullValue = 1 OR VariableLength = 1 THEN 8
                                ELSE 0
                         END +[DataLength]) AS BufferFreeBytes
,      CAST(((32768. % SUM(CASE WHEN NullValue = 1 OR VariableLength = 1 THEN 8
                                ELSE 0
                         END +[DataLength])) / 32768) * 100 AS DECIMAL(8,5)) AS [BufferFreePct]
FROM T
GROUP BY T.[Table_Name]
```

Monitoring DMS

- Use performance monitor to track send and receive queue depth
- Use sys.dm_pdw_dms_workers to monitor throughput of the workers

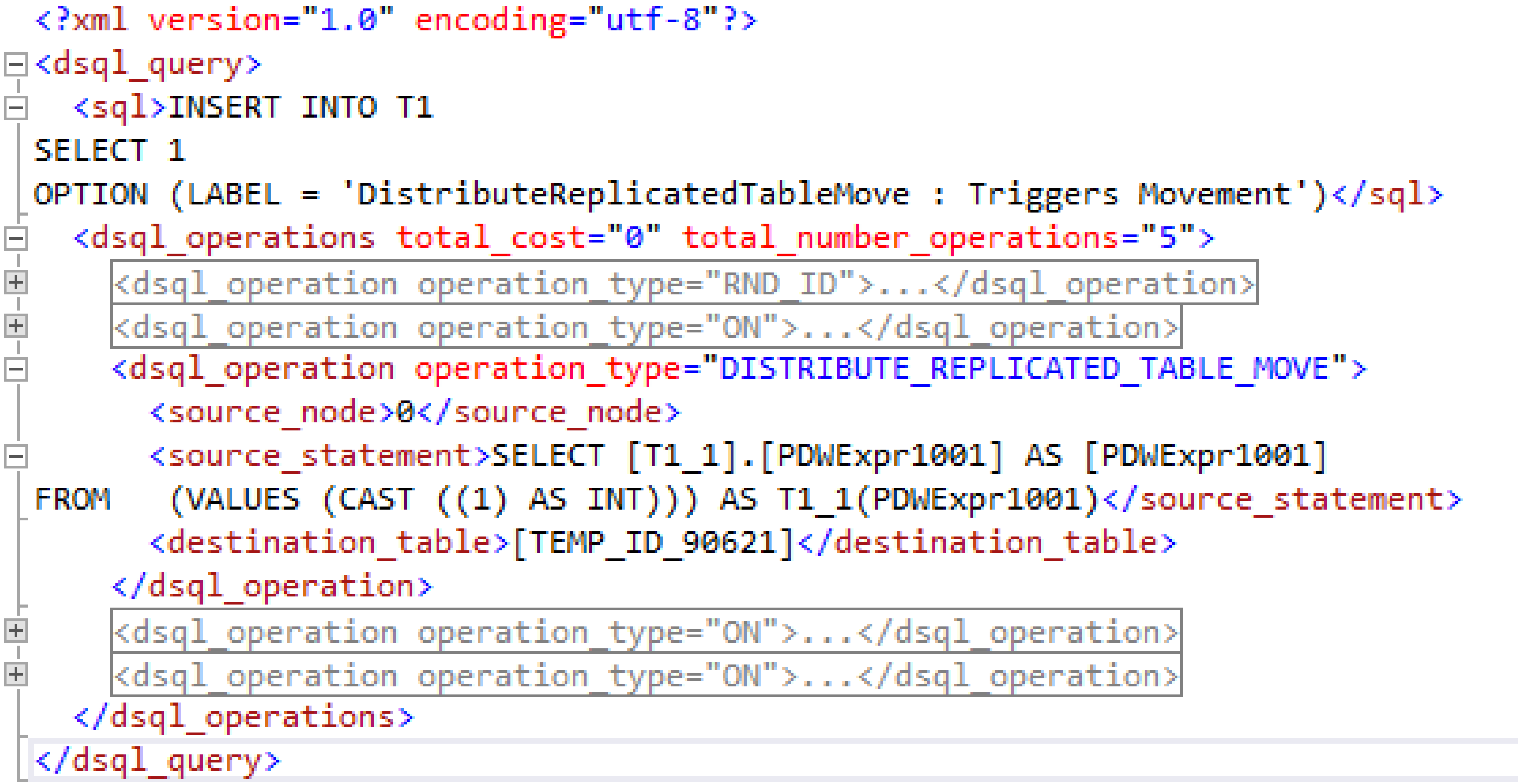


```
SELECT r.request_id         AS Request_request_id         ,      w.dms_step_index     AS Worker_dms_step_index
,      r.session_id         AS Request_session_id         ,      w.pdw_node_id        AS Worker_pdw_node_id
,      r.status             AS Request_status             ,      w.distribution_id    AS Worker_distribution_id
,      r.submit_time        AS Request_submit_time        ,      w.type               AS Worker_type
,      r.start_time         AS Request_start_time         ,      w.status             AS Worker_status
,      r.end_compile_time   AS Request_end_compile_time   ,      w.bytes_per_sec      AS Worker_bytes_per_sec
,      r.end_time           AS Request_end_time           ,      w.bytes_processed    AS Worker_bytes_processed
,      r.total_elapsed_time AS Request_total_elapsed_time ,      w.rows_processed     AS Worker_rows_processed
,      r.[label]            AS Request_label              ,      w.start_time         AS Worker_start_time
,      r.error_id           AS Request_error_id           ,      w.end_time           AS Worker_end_time
,      d.name               AS Request_name               ,      w.total_elapsed_time AS Worker_total_elapsed_time
,      r.command            AS Request_command            ,      w.cpu_time           AS Worker_cpu_time
,      r.resource_class     AS Request_resource_class     ,      w.query_time         AS Worker_query_time
,      s.step_index         AS Step_step_index            ,      w.buffers_available  AS Worker_buffers_available
,      s.operation_type     AS Step_operation_type        ,      w.dms_cpid           AS Worker_dms_cpid
,      s.distribution_type  AS Step_distribution_type     ,      w.sql_spid           AS Worker_sql_spid
,      s.location_type      AS Step_location_type         ,      w.error_id           AS Worker_error_id
,      s.status             AS Step_status                ,      w.source_info        AS Worker_source_info
,      s.error_id           AS Step_error_id              ,      w.destination_info   AS Worker_destination_info
,      s.start_time         AS Step_start_time
,      s.end_time           AS Step_end_time
,      s.total_elapsed_time AS Step_total_elapsed_time
,      s.row_count          AS Step_row_count
,      s.command            AS Step_command
FROM   sys.dm_pdw_exec_requests r
JOIN   sys.databases d            ON  r.database_id = d.database_id
JOIN   sys.dm_pdw_request_steps s ON  r.request_id  = s.request_id
JOIN   sys.dm_pdw_dms_workers w   ON  s.request_id  = w.request_id
                                  AND s.step_index  = w.step_index
WHERE [Label] = 'Move';
```

Single Row Inserts & replicated tables

```
CREATE TABLE T1(col1 INT)
WITH (DISTRIBUTION=REPLICATE)

INSERT INTO T1
SELECT 1
OPTION (LABEL = 'DistributeReplicatedTableMove : Triggers Movement');
```

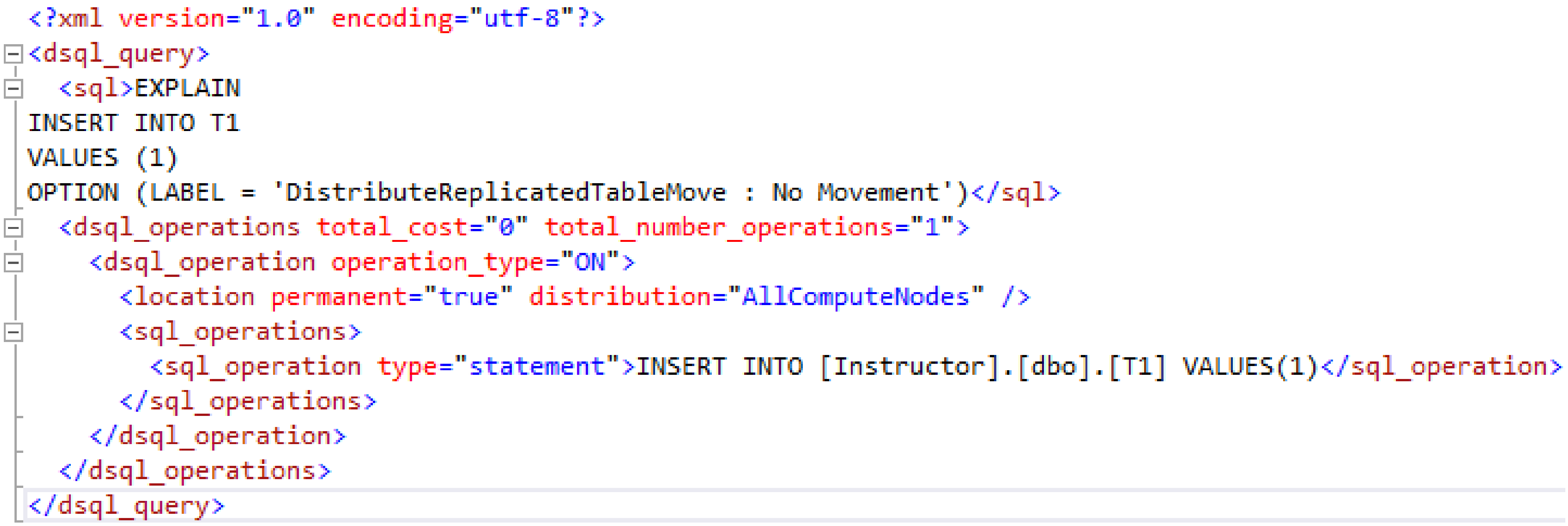
Movement



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>INSERT INTO T1
SELECT 1
OPTION (LABEL = 'DistributeReplicatedTableMove : Triggers Movement')</sql>
  <dsql_operations total_cost="0" total_number_operations="5">
    <dsql_operation operation_type="RND_ID">...</dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="DISTRIBUTE_REPLICATED_TABLE_MOVE">
      <source_node>0</source_node>
      <source_statement>SELECT [T1_1].[PDWExpr1001] AS [PDWExpr1001]
FROM   (VALUES (CAST ((1) AS INT))) AS T1_1(PDWExpr1001)</source_statement>
      <destination_table>[TEMP_ID_90621]</destination_table>
    </dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
    <dsql_operation operation_type="ON">...</dsql_operation>
  </dsql_operations>
</dsql_query>
```

No Movement

```
INSERT INTO T1
VALUES (1)
OPTION (LABEL = 'DistributeReplicatedTableMove : No Movement');
```



```
<?xml version="1.0" encoding="utf-8"?>
<dsql_query>
  <sql>EXPLAIN
INSERT INTO T1
VALUES (1)
OPTION (LABEL = 'DistributeReplicatedTableMove : No Movement')</sql>
  <dsql_operations total_cost="0" total_number_operations="1">
    <dsql_operation operation_type="ON">
      <location permanent="true" distribution="AllComputeNodes" />
      <sql_operations>
        <sql_operation type="statement">INSERT INTO [Instructor].[dbo].[T1] VALUES(1)</sql_operation>
      </sql_operations>
    </dsql_operation>
  </dsql_operations>
</dsql_query>
```

Microsoft

The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.