DAT220x

Delivering a Data Warehouse in the Cloud

Lab 02 | Designing and Querying Data Warehouses

Overview

In this lab, you will create and design tables in Azure SQL Data Warehouse you created in Lab 1. You will work with distributed tables, implement partitioned and replicated tables, apply indexes and statistics to the tables.

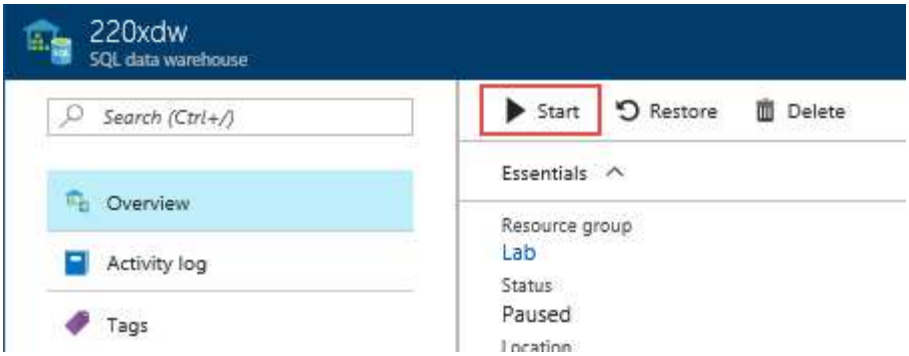
Note: The four labs in this course are cumulative. You cannot complete the following labs if this lab has not been successfully completed.

Exercise 1: Resume the Lab Environment

In this exercise, you will resume your SQL DW from its paused state, and restart your lab virtual machine.

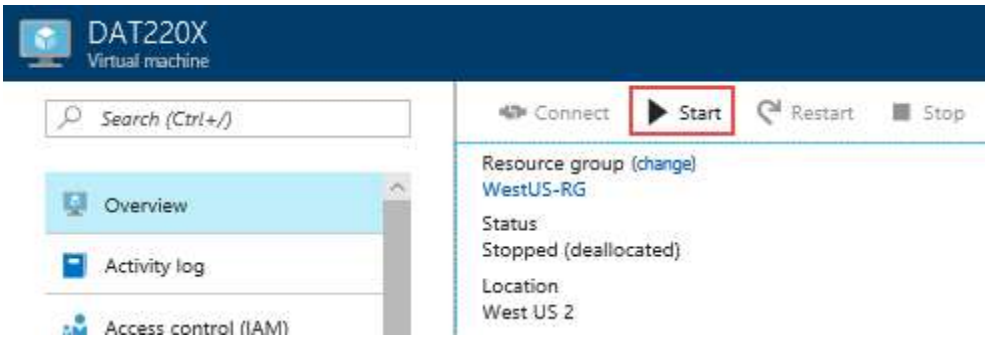
1. Sign in to the Azure Portal by using your subscription. Navigate to the Lab resource group you created in the last lab, then click to open the blade for your SQL DW.

2. If necessary, resume the SQL DW from its paused state:



3. Watch for the notification of the successful resume request.

4. Return to the Lab resource group blade, and click to open the virtual machine you created in your last lab. If necessary, start the VM



5. Once the Connect link becomes available, click Connect and open the Remote Desktop file, as you learned in Lab 1. (You may need to refresh the browser tab.)

6. Using the credentials you created in Lab 1, log in to the VM. Remember to use "use another account" when logging in.

7. Back in the Azure portal, navigate to the SQL DW created in the previous lab.

8. In the Essentials pane, click the Start button to resume your SQL Data Warehouse.

Exercise 2: Creating Tables

In this exercise, you will create two tables, view their properties to look at their default distribution methods and indexes.

1. Use your taskbar shortcut (or other preferred method) to launch SQL Server Management Studio (SSMS).

2. In the Connect to Server dialog, enter the Server name of your SQL DW, using the information in the Portal.

3. Change the Authentication drop-down list to SQL Server Authentication.

4. In the Login and Password boxes, provide the Server admin and password you configured during provisioning of the SQL DW. For your convenience, click Remember password.

5. In the Object Explorer pane, expand the databases node and select the database created in Lab1.

6. On the SSMS toolbar, click the New Querybutton.

7. In the query window, enter the following T-SQL query:

```
IF OBJECT_ID('TableWithoutDistro','U') IS NOT NULL DROP TABLE TableWithoutDistro;
GO
CREATE TABLE [dbo].[TableWithoutDistro]
(    [ProductKey]            int           NOT NULL
,    [OrderDateKey]          int           NOT NULL
,    [CustomerKey]           int           NOT NULL
,    [PromotionKey]          int           NOT NULL
,    [SalesOrderNumber]      nvarchar(20)  NOT NULL
,    [OrderQuantity]         smallint      NOT NULL
,    [UnitPrice]             money         NOT NULL
,    [SalesAmount]           money         NOT NULL
)
;

IF OBJECT_ID('TableWithDistro','U') IS NOT NULL DROP TABLE TableWithDistro;
GO
CREATE TABLE [dbo].[TableWithDistro]
(    [ProductKey]            int           NOT NULL
,    [OrderDateKey]          int           NOT NULL
,    [CustomerKey]           int           NOT NULL
,    [PromotionKey]          int           NOT NULL
,    [SalesOrderNumber]      nvarchar(20)  NOT NULL
,    [OrderQuantity]         smallint      NOT NULL
```

```
,   [UnitPrice]             money       NOT NULL
,   [SalesAmount]           money       NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = ROUND_ROBIN
);
```

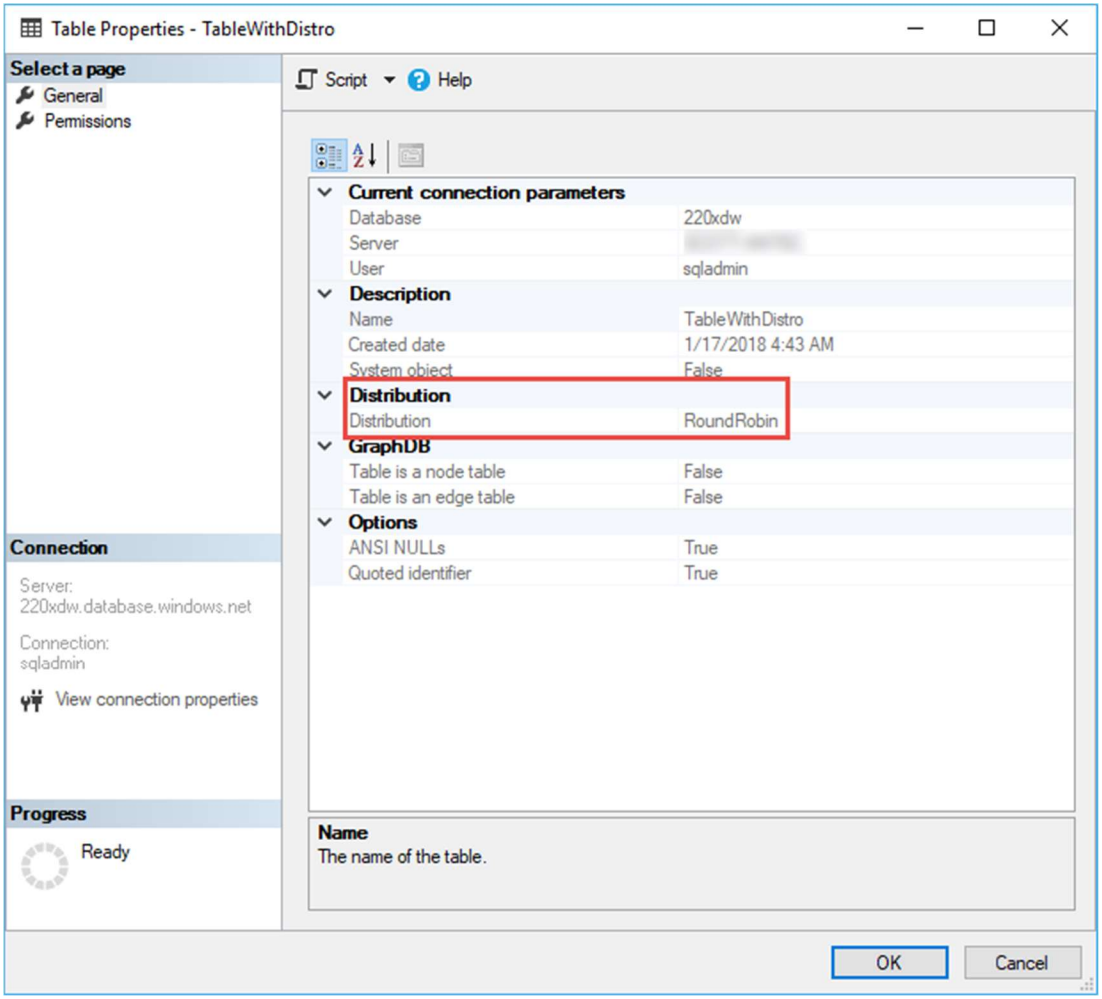
8. Notice the difference between the two CREATE TABLE statements. In the later table, a specific distribution method is specified (Round Robin), whereas in the first statement, no distribution method is specified.

9. Click the Execute button on the toolbar.

10. Once the tables are created, expand the Tables in Object Explorer. If needed, right mouse click the Tables node and select Refresh from the context menu to refresh the table list.

11. In the Object Explorer window in SSMS, right mouse click on the TableWithDistro table and select Properties from the context menu.

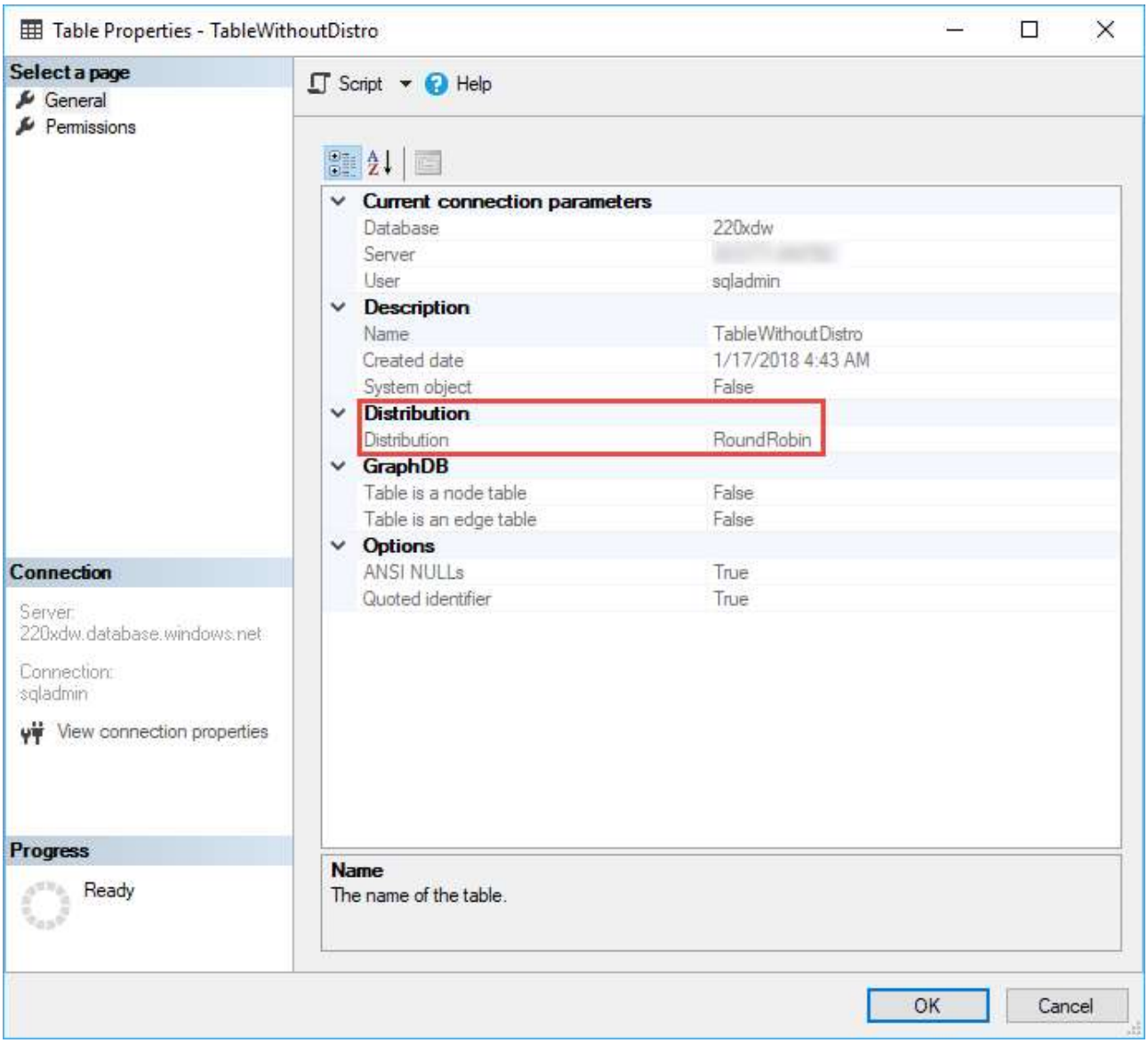
12. On the General page of the Table properties dialog, notice the RoundRobin distribution type. This is expected since the round robin distribution method was specified in the T-SQL statement.



13. Click Cancel on the Table properties dialog.

14. In Object Explorer, right mouse click on the TableWithoutDistro table and select Properties from the context menu.

15. On the General page of the Table properties dialog, notice the distribution type is also RoundRobin. By default, when a distribution method is not specified, the table will be distributed using the round robin distribution method.



16. Click Cancel.

17. In the query window, enter the following T-SQL query:

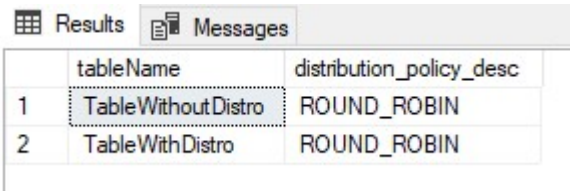
```
SELECT o.name as tableName, distribution_policy_desc
FROM sys.pdw_table_distribution_properties ptdp
```

```
JOIN sys.objects o
ON ptdp.object_id = o.object_id
WHERE ptdp.object_id = object_id('TableWithDistro')
OR ptdp.object_id = object_id('TableWithoutDistro')
```

18. The above query uses the sys.pdw_table_distribution_properties system table, joined with the sys.object system table, to return the distribution method of each table.

19. Click the Execute button on the toolbar.

20. The query results show that both tables have a distribution method of round robin.



	tableName	distribution_policy_desc
1	TableWithoutDistro	ROUND_ROBIN
2	TableWithDistro	ROUND_ROBIN

21. In Object Explorer, right mouse click on the TableWithoutDistro table and select Script table As -> Create To -> New Query Editor Window context menus.

22. A new query window will be created and populated with the T-SQL **CREATE TABLE** statement for the TableWithoutDistro table.

23. By default, SQL Data Warehouse stores a table as a clustered columnstore. Thus, if a clustered columnstore index is not specified during table creation, one is automatically applied. The difference between the CREATE TABLE statement from earlier in this example, and the CREATE TABLE statement generated from SQL Server Management Studio, is that the CREATE TABLE statement for the TableWithoutDistro table from earlier in this example did not contain the parameter to add a clustered columnstore index. Yet, when scripting the table from SSMS, the CREATE TABLE statement includes the columnstore index.

```
CREATE TABLE [dbo].[TableWithoutDistro]
(
    [ProductKey] [int] NOT NULL,
    [OrderDateKey] [int] NOT NULL,
    [CustomerKey] [int] NOT NULL,
    [PromotionKey] [int] NOT NULL,
    [SalesOrderNumber] [nvarchar](20) NOT NULL,
    [OrderQuantity] [smallint] NOT NULL,
    [UnitPrice] [money] NOT NULL,
    [SalesAmount] [money] NOT NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
GO
```

24. Close this query window.

Exercise 3: Working with Distributed Tables

In this exercise, you will create two different tables to understand how distributed methods work and to see how data is distributed using both the round robin and hash distribution methods

1. On the SSMS toolbar, click the New Query button.
2. In the query window, enter the following T-SQL query which creates a simple Orders table with a distribution method of round robin and a clustered columnstore index on the OrderID column. 60 rows are then inserted into the table with exactly the same order date.

```
IF OBJECT_ID('dbo.Orders','U') IS NOT NULL DROP TABLE dbo.Orders;
GO
CREATE TABLE Orders
(
    OrderID int IDENTITY(1,1) NOT NULL
   ,OrderDate datetime NOT NULL
   ,OrderDescription char(15) DEFAULT 'NewOrder'
)
WITH
(   CLUSTERED INDEX (OrderID)
,   DISTRIBUTION = ROUND_ROBIN
);

--Insert rows into table
SET NOCOUNT ON
DECLARE @i INT                  SET @i = 1
DECLARE @date DATETIME SET @date = dateadd(mi,@i,'2017-02-04')
WHILE (@i <= 60)
BEGIN
  INSERT INTO [Orders] (OrderDate) SELECT @date
  SET @i = @i+1;
```

```
END
```

3. Highlight this code in the query window then click the Execute button on the toolbar.

4. Enter the following T-SQL into the query window, then highlight this code and click the Execute button on the toolbar.

```
SELECT
  o.name AS tableName,
  pnp.pdw_node_id,
  pnp.distribution_id,
  pnp.rows
FROM
  sys.pdw_nodes_partitions AS pnp
  JOIN sys.pdw_nodes_tables AS NTables
  ON pnp.object_id = NTables.object_id
  AND pnp.pdw_node_id = NTables.pdw_node_id
  JOIN sys.pdw_table_mappings AS TMap
  ON NTables.name = TMap.physical_name
  AND substring(TMap.physical_name,40, 10) = pnp.distribution_id
  JOIN sys.objects AS o
  ON TMap.object_id = o.object_id
WHERE o.name in ('orders')
ORDER BY distribution_id
```

5. The above query uses some of the SQL DW DMVs to show how data was distributed across the distributions using the round robin distribution method. In the results you can see that the data was distributed somewhat evenly across the 60 distributions. Your results may differ slightly.

Results | Messages

	tableName	pdw_node_id	distribution_id	rows
1	Orders	36	1	0
2	Orders	36	2	2
3	Orders	36	3	3
4	Orders	36	4	1
5	Orders	36	5	1
6	Orders	36	6	2
7	Orders	36	7	0
8	Orders	36	8	1
9	Orders	36	9	0
10	Orders	36	10	1
11	Orders	36	11	0
12	Orders	36	12	2
13	Orders	36	13	2
14	Orders	36	14	0
15	Orders	36	15	1
16	Orders	36	16	1
17	Orders	36	17	4

6. Next, we'll hash distribute a table and view the resulting data distribution.

7. Enter the following T-SQL into the query window, then highlight this code and click the Execute button on the toolbar. This code creates a similar orders table as the previous example, but uses the HASH distribution method on the OrderDate column. The same 60 rows are inserted into the table, again using the same order date for the OrderDate column.

```
CREATE TABLE Orders2
(
    OrderID int IDENTITY(1,1) NOT NULL
   ,OrderDate datetime NOT NULL
   ,OrderDescription char(15) DEFAULT 'NewOrder'
)
WITH
(   CLUSTERED INDEX (OrderID)
,   DISTRIBUTION = HASH(OrderDate)
);


SET NOCOUNT ON
DECLARE @i INT                SET @i = 1
DECLARE @date DATETIME SET @date = dateadd(mi,@i,'2017-02-04')
WHILE (@i <= 60)
BEGIN
  INSERT INTO [Orders2] (OrderDate) SELECT @date
  SET @i = @i+1;
END
```

8. Enter the following T-SQL into the query window, then highlight this code and click the Execute button on the toolbar.

```
SELECT
  o.name AS tableName,
  pnp.pdw_node_id,
  pnp.distribution_id,
  pnp.rows
FROM
  sys.pdw_nodes_partitions AS pnp
  JOIN sys.pdw_nodes_tables AS NTables
  ON pnp.object_id = NTables.object_id
  AND pnp.pdw_node_id = NTables.pdw_node_id
  JOIN sys.pdw_table_mappings AS TMap
  ON NTables.name = TMap.physical_name
  AND substring(TMap.physical_name,40, 10) = pnp.distribution_id
  JOIN sys.objects AS o
  ON TMap.object_id = o.object_id
 WHERE o.name in ('orders2')
```

9. The query results show that all the rows went into the same distribution because the same date was used for each row and the SQL Data Warehouse hash function is deterministic (each row is assigned to one distribution based on the hash).

Results | Messages

	tableName	pdw_node_id	distribution_id	rows
1	Orders2	36	44	0
2	Orders2	36	57	0
3	Orders2	36	7	60
4	Orders2	36	35	0
5	Orders2	36	60	0
6	Orders2	36	41	0
7	Orders2	36	36	0
8	Orders2	36	11	0
9	Orders2	36	56	0
10	Orders2	36	37	0
11	Orders2	36	40	0
12	Orders2	36	26	0
13	Orders2	36	54	0
14	Orders2	36	58	0
15	Orders2	36	43	0
16	Orders2	36	46	0
17	Orders2	36	12	0

10. Best practices recommend using the hash distribution as much as possible to minimize data movement, which will improve query performance.

11. Close this query window.

Exercise 4: Implementing Replicated Tables

In this exercise, you will replicate a table. Replicated tables removes the need to transfer data across compute nodes by replicating a full copy of the data of the specified table to each compute node. The best candidates for replicated tables are tables whose size is less than 2 GB compressed and are small dimension tables. In this exercise, you will convert an existing round-robin table to a replicated table.

All the tables in the sample database use a hash distribution method, so in order to convert a round-robin table to a replicated table, you will first need to convert the table from a hash distribution to a round-robin distribution. This step is necessary to more evenly distribute the data across distributions in order to show data movement.

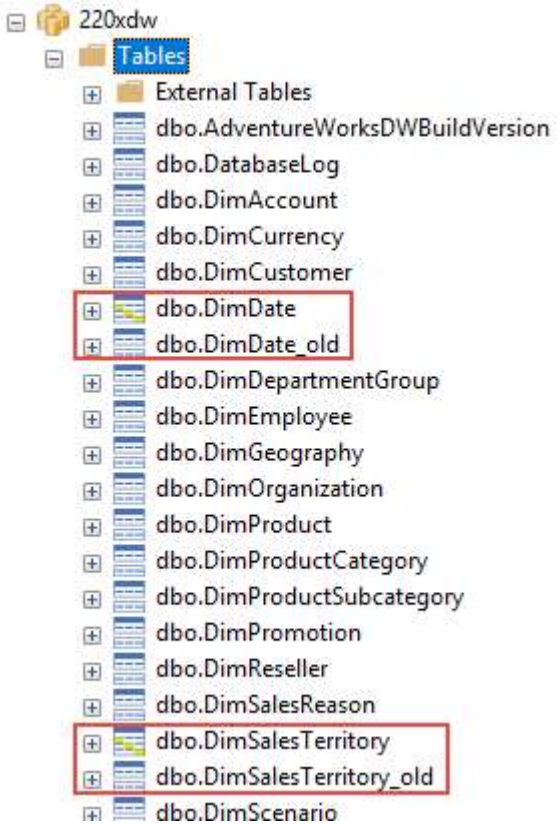
1. In SQL Server Management Studio, in the Object Explorer pane, expand the databases node and select the database created in Lab1.
2. On the SSMS toolbar, click the New Query button.
3. In the query window, enter the following T-SQL which creates two new tables called DimDate_REPLICATE and DimSalesTerritory_REPLICATE. Both of the tables use the ROUND_ROBIN distribution method. The code also renames the original tables, then names the REPLICATE tables to the original table names.

```
CREATE TABLE [dbo].[DimSalesTerritory_REPLICATE]
WITH
  (
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = ROUND_ROBIN
  )
AS SELECT * FROM [dbo].[DimSalesTerritory]
OPTION  (LABEL  = 'CTAS : DimSalesTerritory_REPLICATE')

-- Switch table names
RENAME OBJECT [dbo].[DimSalesTerritory] to [DimSalesTerritory_old];
RENAME OBJECT [dbo].[DimSalesTerritory_REPLICATE] TO [DimSalesTerritory];

CREATE TABLE [dbo].[DimDate_REPLICATE]
WITH
  (
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = ROUND_ROBIN
  )
AS SELECT * FROM [dbo].[DimDate]
OPTION  (LABEL  = 'CTAS : DimDate_REPLICATE')

-- Switch table names
RENAME OBJECT [dbo].[DimDate] to [DimDate_old];
RENAME OBJECT [dbo].[DimDate_REPLICATE] TO [DimDate];
```

4. Click the Execute button.

5. The CREATE TABLE statements in this example uses the feature called CREATE TABLE AS SELECT, or CTAS. This statement is a fully parallelized operation that creates a new table based on the output of the SELECT statement.

6. In Object Explorer, right click the Tables node and select refresh to refresh the table list to see the two new tables.



7. In the query window, enter the following T-SQL, highlight it, then click the Execute button.

```
SELECT o.name as tableName, distribution_policy_desc
FROM sys.pdw_table_distribution_properties ptdp
JOIN sys.objects o
ON ptdp.object_id = o.object_id
WHERE o.name in ('DimDate','DimSalesTerritory','FactInternetSales')
```

8. In the query results, notice that both the DimSalesTerritory and Dim Date tables now have a ROUND_ROBIN distribution method, and the third table, FactInternetSales, uses a HASH distribution method.

9. In the query window, enter the following T-SQL, highlight it, then click the Execute button.

```
SELECT
  o.name AS tableName,
  pnp.pdw_node_id,
  pnp.distribution_id,
  pnp.rows
FROM
  sys.pdw_nodes_partitions AS pnp
  JOIN sys.pdw_nodes_tables AS NTables
```

```
  ON pnp.object_id = NTables.object_id
  AND pnp.pdw_node_id = NTables.pdw_node_id
  JOIN sys.pdw_table_mappings AS TMap
  ON NTables.name = TMap.physical_name
  AND substring(TMap.physical_name,40, 10) = pnp.distribution_id
  JOIN sys.objects AS o
  ON TMap.object_id = o.object_id
WHERE o.name in ('DimSalesTerritory')
 ORDER BY distribution_id
```

10. The query results are similar to those from the previous exercise. Specifying the DimSalesTerritory, the query results will tell you how the data was distributed across the distributions when converting the table from a HASH distribution method to a ROUND_ROBIN distribution method.

11. Replace DimSalesTerritory with DimDate and re-execute the query to see how that data was distributed using the ROUND_ROBIN distribution method.

12. In the query window, enter the following T-SQL, highlight it, then click the Execute button. Notice that this query joins the three tables mentioned above; DimDate, DimSalesTerritory, and FactInternetSales.

```
SELECT [TotalSalesAmount] = SUM(SalesAmount)
FROM dbo.FactInternetSales s
INNER JOIN dbo.DimDate d
  ON d.DateKey = s.OrderDateKey
INNER JOIN dbo.DimSalesTerritory t
  ON t.SalesTerritoryKey = s.SalesTerritoryKey
WHERE d.FiscalYear = 2004
  AND t.SalesTerritoryGroup = 'North America'
OPTION (LABEL = 'STATEMENT:RoundRobinQuery');
```

13. The above query returns the total sales in North America for the fiscal year 2004. The query implements a concept called Query Labels which is helpful for tracking down problem queries.

14. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This query uses the sys.dm_pdw_exec_requests DMV which contains information about all requests currently or recently active in SQL DW, and the sys.dm_pdw_request_steps DMV, which contains information about all the steps that make up a given request or query.

```
SELECT step_index,
     operation_type
FROM   sys.dm_pdw_exec_requests er
JOIN   sys.dm_pdw_request_steps rs
     ON er.request_id = rs.request_id
WHERE  er.[label] = 'STATEMENT:RoundRobinQuery';
```

15. The query results reveal multiple move operations (indicating data movement) as specified by the operation type ‘BroadcastMoveOperation’. Since DimDate and DimSalesTerritory use round-robin distribution, a join copies the data of each table in full to each Compute node.

Results | Messages

	step_index	operation_type
1	0	RandomIDOperation
2	1	OnOperation
3	2	BroadcastMoveOperation
4	3	RandomIDOperation
5	4	OnOperation
6	5	BroadcastMoveOperation
7	6	OnOperation
8	7	PartitionMoveOperation
9	8	OnOperation
10	9	OnOperation
11	10	ReturnOperation
12	11	OnOperation
13	0	RandomIDOperation
14	1	OnOperation
15	2	BroadcastMoveOperation
16	3	RandomIDOperation
17	4	OnOperation
18	5	BroadcastMoveOperation

16. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This query creates two new tables for DimDate and DimSalesTerritory, both with a distribution method of REPLICATE. Using the CTAS function, the data is copied from the existing DimDate and DimSalesTerritory tables, then the tables are renamed.

```
CREATE TABLE [dbo].[DimSalesTerritory_REPLICATE]
WITH
  (
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = REPLICATE
  )
AS SELECT * FROM [dbo].[DimSalesTerritory]
OPTION  (LABEL  = 'CTAS : DimSalesTerritory_REPLICATE')

-- Switch table names
RENAME OBJECT [dbo].[DimSalesTerritory] to [DimSalesTerritory_RR];
RENAME OBJECT [dbo].[DimSalesTerritory_REPLICATE] TO [DimSalesTerritory];

CREATE TABLE [dbo].[DimDate_REPLICATE]
WITH
  (
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = REPLICATE
  )
AS SELECT * FROM [dbo].[DimDate]
OPTION  (LABEL  = 'CTAS : DimDate_REPLICATE')

-- Switch table names
RENAME OBJECT [dbo].[DimDate] to [DimDate_RR];
RENAME OBJECT [dbo].[DimDate_REPLICATE] TO [DimDate];
```

17. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This query is similar to the query in step 12, but a new query label has been used to be able to identify this query when querying the DMVs.

```
SELECT [TotalSalesAmount] = SUM(SalesAmount)
FROM dbo.FactInternetSales s
INNER JOIN dbo.DimDate d
  ON d.DateKey = s.OrderDateKey
INNER JOIN dbo.DimSalesTerritory t
  ON t.SalesTerritoryKey = s.SalesTerritoryKey
WHERE d.FiscalYear = 2004
  AND t.SalesTerritoryGroup = 'North America'
OPTION (LABEL = 'STATEMENT:ReplicatedTableQuery');
```

18. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This query is similar to the query in step 14, but uses the query label used in the query in the previous step.

```
SELECT step_index,
     operation_type
FROM   sys.dm_pdw_exec_requests er
JOIN   sys.dm_pdw_request_steps rs
     ON er.request_id = rs.request_id
WHERE  er.[label] = 'STATEMENT:ReplicatedTableQuery';
```

19. The query results still reveal move operations. This is because the first time the query is executed (with the table having a REPLICATE distribution method), the data is replicated to the other Compute nodes.

Results | Messages

	step_index	operation_type
1	0	OnOperation
2	1	OnOperation
3	2	OnOperation
4	3	BroadcastMoveOperation
5	4	OnOperation
6	5	OnOperation
7	6	OnOperation
8	7	OnOperation
9	8	OnOperation
10	9	BroadcastMoveOperation
11	10	OnOperation
12	11	OnOperation
13	12	RandomIDOperation
14	13	OnOperation
15	14	BroadcastMoveOperation

20. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This query is similar to the query in step 17, but uses a new query label.

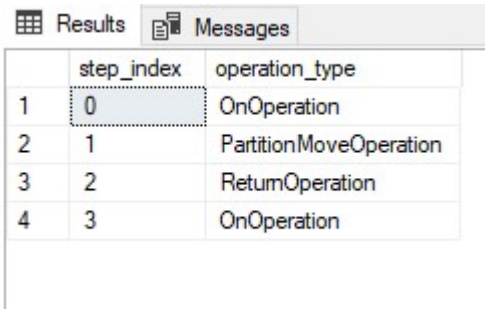
```
SELECT [TotalSalesAmount] = SUM(SalesAmount)
```

```
FROM dbo.FactInternetSales s
INNER JOIN dbo.DimDate d
  ON d.DateKey = s.OrderDateKey
INNER JOIN dbo.DimSalesTerritory t
  ON t.SalesTerritoryKey = s.SalesTerritoryKey
WHERE d.FiscalYear = 2004
  AND t.SalesTerritoryGroup = 'North America'
 OPTION (LABEL = 'STATEMENT:ReplicatedTableQuery2');
```

21. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This query is similar to the query in step 18, but uses the query label.

```
SELECT step_index,
     operation_type
FROM   sys.dm_pdw_exec_requests er
JOIN   sys.dm_pdw_request_steps rs
     ON er.request_id = rs.request_id
WHERE  er.[label] = 'STATEMENT:ReplicatedTableQuery2';
```

22. Running the query again reveals no BroadcastMoveOperations, indicating that no more data movement.



Results | Messages

	step_index	operation_type
1	0	OnOperation
2	1	PartitionMoveOperation
3	2	ReturnOperation
4	3	OnOperation

23. Close this query window.

Exercise 5: Managing Statistics

In this exercise, you will add statistics to the tables created in the previous exercise. Applying statistics is one of the most critical things you can do to optimize your queries and improve query performance, as it tells the query optimizer about your data so it can generate better query plans. Statistics are not automatically created so they must be created and updated manually.

1. On the SSMS toolbar, click the New Query button.

2. In the query window, enter the following T-SQL and click the Execute button. This query uses several system tables to determine the last statistics update for the DimDate and DimSalesTerritory tables.

```
SELECT
    tb.[name] AS [table_name],
    co.[name] AS [column_name],
```

```
    STATS_DATE(st.[object_id],st.[stats_id]) AS [stats_last_updated_date]
FROM
    sys.objects ob
    JOIN sys.stats st ON ob.[object_id] = st.[object_id]
    JOIN sys.stats_columns sc ON st.[stats_id] = sc.[stats_id]
        AND st.[object_id] = sc.[object_id]
    JOIN sys.columns co ON sc.[column_id] = co.[column_id]
        AND sc.[object_id] = co.[object_id]
    JOIN sys.types  ty ON co.[user_type_id] = ty.[user_type_id]
    JOIN sys.tables tb ON co.[object_id] = tb.[object_id]
WHERE
    st.[user_created] = 1
AND tb.[name] IN ('DimDate', 'DimSalesTerritory');
```

3. The query results reveal that statistics have not been created on these two tables, as shown via a NULL value in the stats_last_updated_date column.

Results | Messages

	table_name	column_name	stats_last_updated_date
1	DimDate	DateKey	NULL
2	DimDate	FullDateAlternateKey	NULL
3	DimDate	DayNumberOfWeek	NULL
4	DimDate	EnglishDayNameOfWeek	NULL
5	DimDate	SpanishDayNameOfWeek	NULL
6	DimDate	FrenchDayNameOfWeek	NULL
7	DimDate	FiscalSemester	NULL
8	DimDate	MonthNumberOfYear	NULL
9	DimDate	CalendarQuarter	NULL
10	DimDate	CalendarYear	NULL
11	DimDate	CalendarSemester	NULL
12	DimDate	FiscalQuarter	NULL
13	DimDate	FiscalYear	NULL
14	DimDate	DayNumberOfMonth	NULL
15	DimDate	DayNumberOfYear	NULL
16	DimDate	WeekNumberOfYear	NULL
17	DimDate	EnglishMonthName	NULL

4. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This T-SQL creates statistics for all the columns in both tables.

```
    CREATE STATISTICS [SalesTerritoryKey] ON [DimSalesTerritory] ([SalesTerritoryKey]);
    CREATE STATISTICS [SalesTerritoryAlternateKey] ON [DimSalesTerritory]
([SalesTerritoryAlternateKey]);
    CREATE STATISTICS [SalesTerritoryRegion] ON [DimSalesTerritory]
([SalesTerritoryRegion]);
    CREATE STATISTICS [SalesTerritoryCountry] ON [DimSalesTerritory]
([SalesTerritoryCountry]);
    CREATE STATISTICS [SalesTerritoryGroup] ON [DimSalesTerritory] ([SalesTerritoryGroup]);

    CREATE STATISTICS [DateKey] ON DimDate ([DateKey]);
    CREATE STATISTICS [FullDateAlternateKey] ON DimDate ([FullDateAlternateKey]);
    CREATE STATISTICS [DayNumberOfWeek] ON DimDate ([DayNumberOfWeek]);
    CREATE STATISTICS [EnglishDayNameOfWeek] ON DimDate ([EnglishDayNameOfWeek]);
```

```
CREATE STATISTICS [SpanishDayNameOfWeek] ON DimDate ([SpanishDayNameOfWeek]);
CREATE STATISTICS [FrenchDayNameOfWeek] ON DimDate ([FrenchDayNameOfWeek]);
CREATE STATISTICS [DayNumberOfMonth] ON DimDate ([DayNumberOfMonth]);
CREATE STATISTICS [DayNumberOfYear] ON DimDate ([DayNumberOfYear]);
CREATE STATISTICS [WeekNumberOfYear] ON DimDate ([WeekNumberOfYear]);
CREATE STATISTICS [EnglishMonthName] ON DimDate ([EnglishMonthName]);
CREATE STATISTICS [SpanishMonthName] ON DimDate ([SpanishMonthName]);
CREATE STATISTICS [FrenchMonthName] ON DimDate ([FrenchMonthName]);
CREATE STATISTICS [MonthNumberOfYear] ON DimDate ([MonthNumberOfYear]);
CREATE STATISTICS [CalendarQuarter] ON DimDate ([CalendarQuarter]);
CREATE STATISTICS [CalendarYear] ON DimDate ([CalendarYear]);
CREATE STATISTICS [CalendarSemester] ON DimDate ([CalendarSemester]);
CREATE STATISTICS [FiscalQuarter] ON DimDate ([FiscalQuarter]);
CREATE STATISTICS [FiscalYear] ON DimDate ([FiscalYear]);
CREATE STATISTICS [FiscalSemester] ON DimDate ([FiscalSemester]);
```

5. In the query window, highlight and execute the query in Step 2. The query results show that statistics have now been created for both tables.

Results | Messages

	table_name	column_name	stats_last_updated_date
30	DimDate	DateKey	2018-01-19 07:00:43.380
31	DimDate	FullDateAlternateKey	2018-01-19 07:00:44.660
32	DimDate	DayNumberOfWeek	2018-01-19 07:00:45.927
33	DimDate	EnglishDayNameOfWeek	2018-01-19 07:00:47.130
34	DimDate	SpanishDayNameOfWeek	2018-01-19 07:00:49.707
35	DimDate	FrenchDayNameOfWeek	2018-01-19 07:00:51.020
36	DimDate	DayNumberOfMonth	2018-01-19 07:00:52.300
37	DimDate	DayNumberOfYear	2018-01-19 07:00:53.520
38	DimDate	WeekNumberOfYear	2018-01-19 07:00:54.707
39	DimDate	EnglishMonthName	2018-01-19 07:00:56.033
40	DimDate	SpanishMonthName	2018-01-19 07:00:57.333
41	DimDate	FrenchMonthName	2018-01-19 07:00:58.503
42	DimDate	MonthNumberOfYear	2018-01-19 07:00:59.770
43	DimDate	CalendarQuarter	2018-01-19 07:01:01.037
44	DimDate	CalendarYear	2018-01-19 07:01:02.393
45	DimDate	CalendarSemester	2018-01-19 07:01:03.690
46	DimDate	FiscalQuarter	2018-01-19 07:01:05.003
47	DimDate	FiscalYear	2018-01-19 07:01:06.457
48	DimDate	FiscalSemester	2018-01-19 07:01:07.660

6. Best practice states that statistics be kept up to date as new rows are added to the table. The following T-SQL statement updates the statistics column for the DateKey column in the DimDate table.

```
UPDATE STATISTICS [dbo].[DimDate] ([DateKey]);
```

7. To following T-SQL statement updates the statistics for all the columns in the DimDate table.

```
UPDATE STATISTICS [dbo].[DimDate]
```

8. Close this query window.

Exercise 6: Implementing Partitions

In this exercise, you will create a partitioned table and switch data in and out of a partition. Partitioning can have benefits to both data maintenance, such as speeding up the loading and archiving of data, and query performance, enabling the query optimizer to only access relevant partitions.

1. In SQL Server Management Studio, in the Object Explorer pane, expand the databases node and select the database created in Lab1.
2. On the SSMS toolbar, click the New Query button.
3. In the query window, enter the following T-SQL and click the Execute button. This query creates an OrdersPartition table, creating 8 partitions, partitioning the data on the OrderDate column. Data is then inserted into the table.

```
IF OBJECT_ID('dbo.OrdersPartition','U') IS NOT NULL DROP TABLE dbo.OrdersPartition;
GO
CREATE TABLE OrdersPartition
(
    OrderID int IDENTITY(1,1) NOT NULL
   ,OrderDate datetime NOT NULL
   ,OrderDescription char(15) DEFAULT 'NewOrder'
)
WITH
(
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = ROUND_ROBIN,
    PARTITION
         (
                OrderDate RANGE RIGHT FOR VALUES
                (
                   '2017-02-05T00:00:00.000'
                 , '2017-02-12T00:00:00.000'
                 , '2017-02-19T00:00:00.000'
                 , '2017-02-26T00:00:00.000'
                 , '2017-03-05T00:00:00.000'
                 , '2017-03-12T00:00:00.000'
                 , '2017-03-19T00:00:00.000'
                )
         )
);
GO

SET NOCOUNT ON
DECLARE @i INT                SET @i = 1
DECLARE @date DATETIME SET @date = dateadd(mi,@i,'2017-02-05')
WHILE (@i <= 10)
BEGIN
  INSERT INTO [OrdersPartition] (OrderDate) SELECT @date
  INSERT INTO [OrdersPartition] (OrderDate) SELECT dateadd(week,1,@date)
```

```
  INSERT INTO [OrdersPartition] (OrderDate) SELECT dateadd(week,2,@date)
  INSERT INTO [OrdersPartition] (OrderDate) SELECT dateadd(week,3,@date)
  INSERT INTO [OrdersPartition] (OrderDate) SELECT dateadd(week,4,@date)
  INSERT INTO [OrdersPartition] (OrderDate) SELECT dateadd(week,5,@date)
  SET @i = @i+1;
END

SELECT COUNT(*) FROM OrdersPartition
```

4. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This T-SQL returns the number of rows by partition.

```
SELECT
  o.name AS Table_name,
  pnp.partition_number AS Partition_number,
  sum(pnp.rows) AS Row_count
FROM sys.pdw_nodes_partitions AS pnp
  JOIN sys.pdw_nodes_tables AS NTables ON pnp.object_id = NTables.object_id
        AND pnp.pdw_node_id = NTables.pdw_node_id
  JOIN sys.pdw_table_mappings AS TMap ON NTables.name = TMap.physical_name
        AND substring(TMap.physical_name,40, 10) = pnp.distribution_id
  JOIN sys.objects AS o ON TMap.object_id = o.object_id
WHERE o.name in ('OrdersPartition')
GROUP BY partition_number, o.name, pnp.data_compression_desc;
```

5. The query results show the 8 partitions of the OrdersPartition table and the rows in each partition.

Results | Messages

	Table_name	Partition_number	Row_count
1	OrdersPartition	1	0
2	OrdersPartition	2	10
3	OrdersPartition	3	10
4	OrdersPartition	4	10
5	OrdersPartition	5	10
6	OrdersPartition	6	10
7	OrdersPartition	7	10
8	OrdersPartition	8	0

6. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This T-SQL creates a second table called Orders_Staging which will be used to switch data out of a partition in the production OrdersPartition table.

```
IF OBJECT_ID('dbo.Orders_Staging','U') IS NOT NULL DROP TABLE dbo.Orders_Staging;
GO
CREATE TABLE dbo.Orders_Staging
  (OrderID int IDENTITY(1,1) NOT NULL
  ,OrderDate datetime NOT NULL
```

```
  ,OrderDescription char(15) DEFAULT 'NewOrder'
  )
GO
```

7. In the query window, enter the following T-SQL, highlight it, then click the Execute button.

```
SELECT
  o.name AS Table_name,
  pnp.partition_number AS Partition_number,
  sum(pnp.rows) AS Row_count
FROM sys.pdw_nodes_partitions AS pnp
  JOIN sys.pdw_nodes_tables AS NTables ON pnp.object_id = NTables.object_id
        AND pnp.pdw_node_id = NTables.pdw_node_id
  JOIN sys.pdw_table_mappings AS TMap ON NTables.name = TMap.physical_name
        AND substring(TMap.physical_name,40, 10) = pnp.distribution_id
  JOIN sys.objects AS o ON TMap.object_id = o.object_id
WHERE o.name in ('Orders_Staging')
GROUP BY partition_number, o.name, pnp.data_compression_desc;
```

8. The query results show the default partition with no rows.

Results | Messages

	Table_name	Partition_number	Row_count
1	Orders_Staging	1	0

9. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This query switches out the data out of partition 3 of the OrdersPartition table into the default partion of the Orders_Staging table.

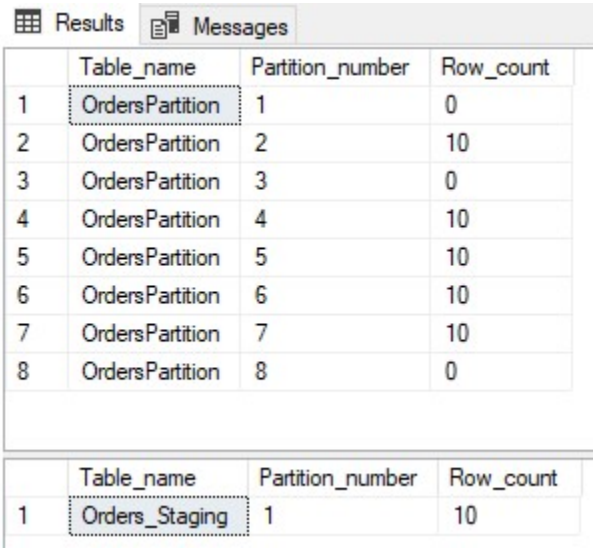
```
ALTER TABLE dbo.OrdersPartition SWITCH PARTITION 3 to dbo.Orders_Staging
```

10. In the query window, enter the following T-SQL, highlight it, then click the Execute button. This query executes the same query in steps 4 and 7, looking at the partition information for both tables.

```
SELECT
  o.name AS Table_name,
  pnp.partition_number AS Partition_number,
  sum(pnp.rows) AS Row_count
FROM sys.pdw_nodes_partitions AS pnp
  JOIN sys.pdw_nodes_tables AS NTables ON pnp.object_id = NTables.object_id
        AND pnp.pdw_node_id = NTables.pdw_node_id
  JOIN sys.pdw_table_mappings AS TMap ON NTables.name = TMap.physical_name
```

```
        AND substring(TMap.physical_name,40, 10) = pnp.distribution_id
  JOIN sys.objects AS o ON TMap.object_id = o.object_id
WHERE o.name in ('OrdersPartition')
GROUP BY partition_number, o.name, pnp.data_compression_desc;

SELECT
  o.name AS Table_name,
  pnp.partition_number AS Partition_number,
  sum(pnp.rows) AS Row_count
FROM sys.pdw_nodes_partitions AS pnp
  JOIN sys.pdw_nodes_tables AS NTables ON pnp.object_id = NTables.object_id
        AND pnp.pdw_node_id = NTables.pdw_node_id
  JOIN sys.pdw_table_mappings AS TMap ON NTables.name = TMap.physical_name
        AND substring(TMap.physical_name,40, 10) = pnp.distribution_id
  JOIN sys.objects AS o ON TMap.object_id = o.object_id
WHERE o.name in ('Orders_Staging')
GROUP BY partition_number, o.name, pnp.data_compression_desc;
```

11. The query results reveal that the 10 rows in partition 3 in the OrdersPatition table have moved, or switched, to the default partition in the Orders_Staging table.



Results | Messages

	Table_name	Partition_number	Row_count
1	OrdersPartition	1	0
2	OrdersPartition	2	10
3	OrdersPartition	3	0
4	OrdersPartition	4	10
5	OrdersPartition	5	10
6	OrdersPartition	6	10
7	OrdersPartition	7	10
8	OrdersPartition	8	0

	Table_name	Partition_number	Row_count
1	Orders_Staging	1	10

12. Close this query window.

You have now completed the lab. Be sure to complete the Finishing Up exercise to shut down and stop the VM, and to pause the SQL DW.

Finishing Up

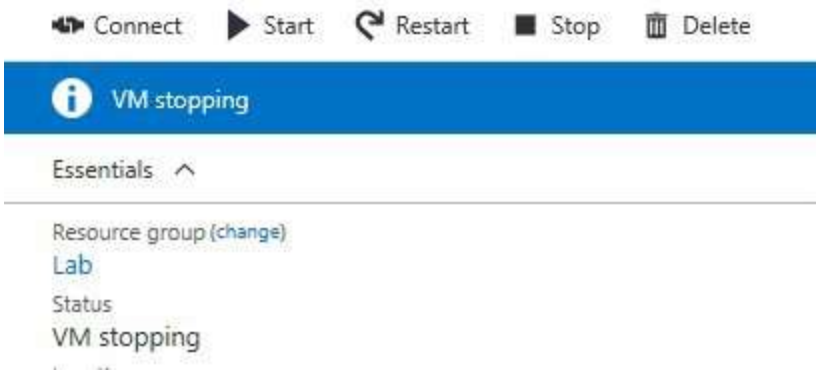
In this task, you will shut down and stop the VM, and pause the SQL DW. If you are immediately continuing to further labs in this course, you can skip this task. However, costs will continue to be incurred by the VM and SQL DW until they are deallocated or paused using the steps below.

1. Close all open applications.

2. Right-click on the Windows button in the bottom left corner of your screen. Click on Shut down or sign out. Be sure this is the menu for you VM and not your desktop!

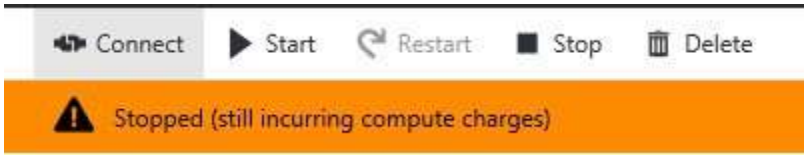
3. Click shut down.

4. In the Azure Portal Web browser page on your desktop, wait until the status of the VM updates.

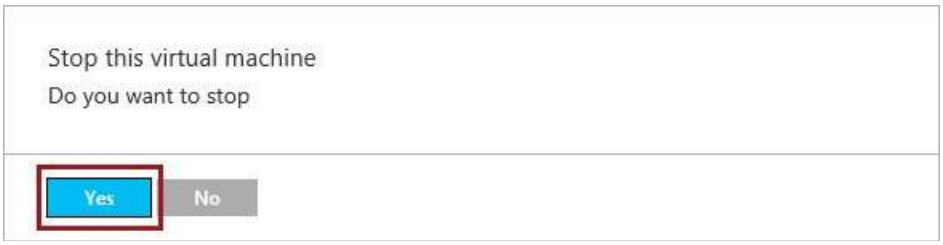


In this state, however, the VM is still billable.

5. To deallocate the VM, click Stop.



6. When prompted to stop the VM, click Yes.



The deallocation can take several minutes to complete.

7. Verify that the VM status updates to Stopped (Deallocated).

In this state, the VM is now not billable.

Note that a deallocated VM will likely acquire a different IP address the next time it is started.

8. Browse the Azure Portal to open the blade for your SQL DW.
9. Click the Pause button. Watch the status change, and confirm that the SQL DW has paused before you finish the lab.
10. Sign out of the Azure Portal.