Microsoft

Rock Your Data

Azure Cloud Data Warehouse

in a Day

Microsoft Azure

https://rockyourdata.cloud | hello@rockyourdata.cloud
Vancouver | Victoria | Calgary | Toronto | Winnipeg

Contents

Lab Prerequisites

You should have a laptop. Before coming to the workshop please make sure you've installed the following on your machine:

a. Windows
   - Azure Data Studio
   - Microsoft Azure Storage Explorer (optional)
   - Visual Studio Community 2017 (optional)
   - SQL Server Management Studio (optional)
   - Power BI Desktop

b. Mac OS
   - Azure Data Studio
   - Microsoft Azure Storage Explorer

Overview

Introduction

In this course, you will learn concepts, strategies, and best practices for designing a cloud-based data warehousing solution using Microsoft Azure SQL Data Warehouse, the petabyte-scale data warehouse in Azure. We will demonstrate how to collect, store, and prepare data for the data warehouse by using other Azure services. We will also explore how to use business intelligence (BI) and ETL tools to perform analysis on your data and integration. This course will help you to understand the fundamental principles of Cloud Analytics and learn core functionality of Azure SQL DW and Azure Data Platform ecosystem.

Data Set

We will use the following data sets:

- Microsoft SQL Server AdventureWorks database that is deploying with Azure SQL Data Warehouse
- Sample TXT file with sales information
- Sample logs files

Course Outline

We will have four labs during workshop. We will cover the following topics:

- Getting started with Azure SQL DW - provision cluster of DW and connecting it with SQL client.
- Learn about Azure SQL DW design principles and available distribution methods as well as other techniques that helps to optimize performance of DW.
- Try Azure Data Factory and will load data into DW as well as use Polybase and create external table.
- We will use Power BI desktop and connect Azure SQL DW.

Credits

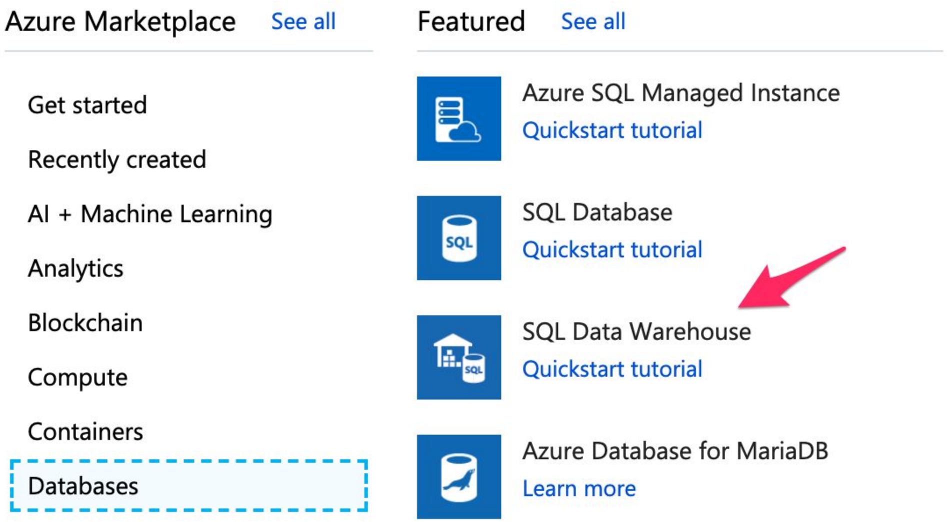
Course was prepared based on technical knowledge of Rock Your Data team in Cloud Data Warehousing, Microsoft documentation, Microsoft Training Materials and online Microsoft courses on Edx.org. Special thank you to Andrew Boudreau for organization and Azure budget.

Lab 01: Provision Azure DW

In this lab, we will provision Azure SQL Data Warehouse (DW) using Microsoft Azure Portal. Moreover, we will connect Azure SQL DW using Azure Data Studio. Finally, we will run SQL queries.

Exercise 1: Launch Azure SQL DW

1. Login to Azure Portal https://portal.azure.com using credentials from Microsoft.
2. On the left bar navigate to **All resources**
3. Click +Add in order to add new resource and choose SQL Data Warehouse



4. We should configure our DW. This table below has the information that we should use for this lab:

Project details	
Subscription	Microsoft Azure Sponsorship 2
Resource group	Cloud-dw-in-a-day
Data warehouse details	
Data warehouse name	clouddw-<your initials>

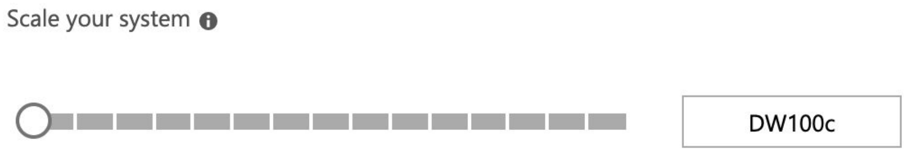
Also, we should create a new server. It gives us a connection string and credentials. Click **Create new** and fill the properties:

Server Name	clouddw-server-<your initials>
Server admin login	clouddw-admin
Password	<your password>
Confirm password	<your password>
Location*	(US) East US

Mark the “Allow Azure services to access the server” and click Ok.
**In Canada there are 2 Microsft Azure regions - Central and East.*

For the Performance level, we will use the smallest DW - DW100c. It is a single node with 60 distributions on it. Microsoft recommends start small and monitor the DW and scale if need. Navigate to **Select performance level.**

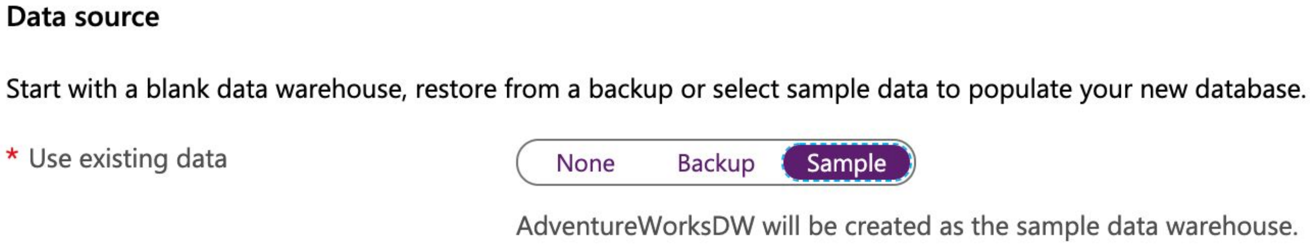
Azure SQL Data Warehouse CPU, memory, and IO are bundled into units of compute scale called Data Warehouse Units (DWUs). A DWU represents an abstract, normalized measure of computing resources and performance. A change to your service level alters the number of DWUs that are available to the system, which in turn adjusts the performance, and the cost, of your system.



Scale your system

DW100c

Then click Additional settings. We will use Sample data:



Data source

Start with a blank data warehouse, restore from a backup or select sample data to populate your new database.

* Use existing data — None | Backup | [✓ Sample]

AdventureWorksDW will be created as the sample data warehouse.

There is also an option for Collations. With Sample data, we won’t change anything.

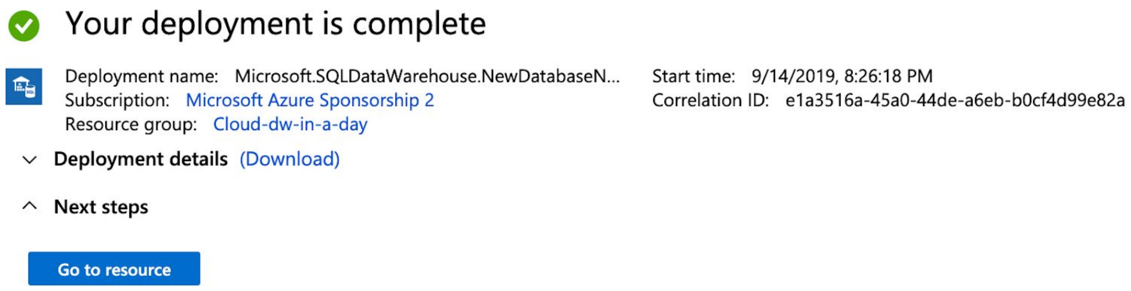
Collations provide the locale, code page, sort order and character sensitivity rules for character-based data types. Once chosen, all columns and expressions requiring collation information inherit the chosen collation from the database setting.

Move to the Tags and specify:

- Name - workshop
- Value - clouddw

Finally, go to the **Review + create** section and click **Create**. Watch for the notification of the successful deployment.

5. After deployment of DW we should click on Go to resource button:

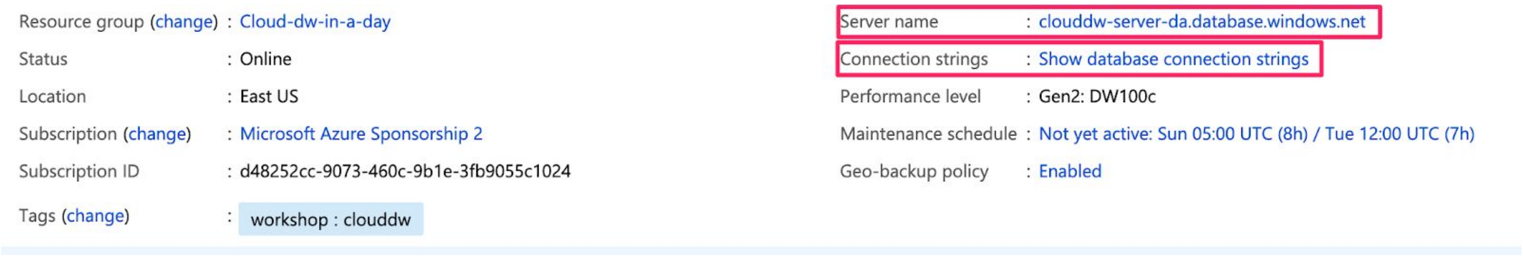


Your deployment is complete

Deployment name: Microsoft.SQLDataWarehouse.NewDatabaseN...
Subscription: Microsoft Azure Sponsorship 2
Resource group: Cloud-dw-in-a-day

Start time: 9/14/2019, 8:26:18 PM
Correlation ID: e1a3516a-45a0-44de-a6eb-b0cf4d99e82a

6. You can overview information about your DW and available options. Our next goal is to connect Azure SQL DW with SQL client. It could be Azure Data Studio or any other SQL client. You need to get your server name or you can copy connection string in case if you want to connect with JDBC or ODBC drivers.

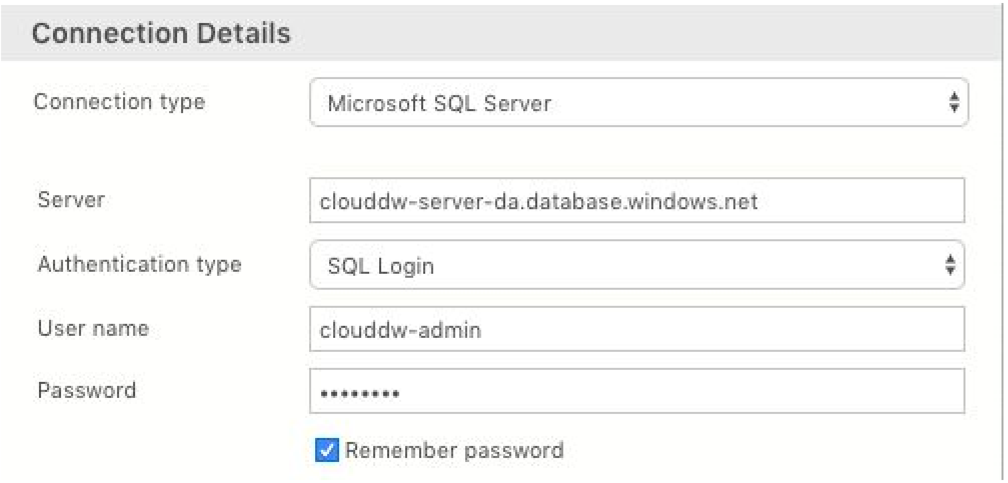


Resource group (change)	: Cloud-dw-in-a-day	Server name	: clouddw-server-da.database.windows.net
Status	: Online	Connection strings	: Show database connection strings
Location	: East US	Performance level	: Gen2: DW100c
Subscription (change)	: Microsoft Azure Sponsorship 2	Maintenance schedule	: Not yet active: Sun 05:00 UTC (8h) / Tue 12:00 UTC (7h)
Subscription ID	: d48252cc-9073-460c-9b1e-3fb9055c1024	Geo-backup policy	: Enabled
Tags (change)	: workshop : clouddw		

For example, the server name is **clouddw-server-da.database.windows.net**

Exercise 2: Connecting Azure SQL DW with SQL client

7. Open Azure Data Studio. If you didn’t download it when you can download and install it. Moreover, you can use SQL Server Management Studio.
8. Add a new Connection. We should specify the Server name, user name, and password from step (4).

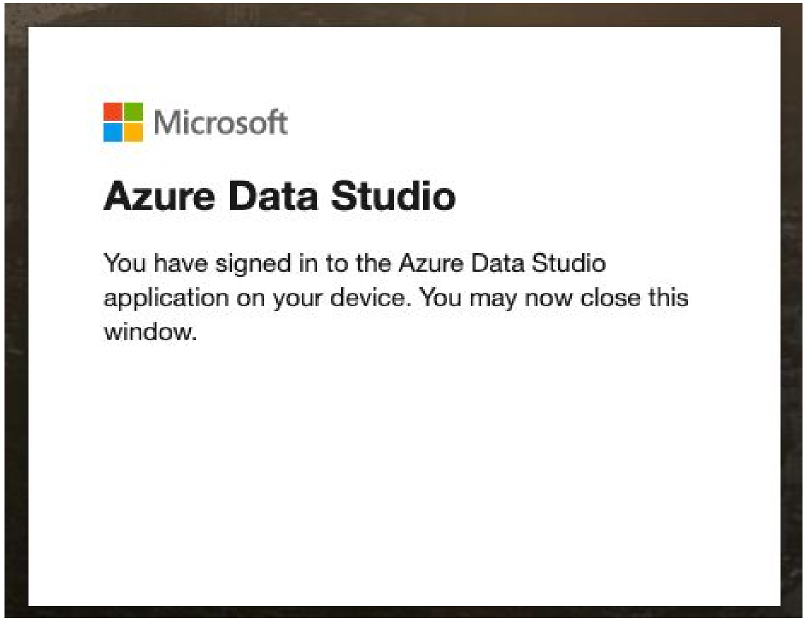


Connection Details

Connection type	Microsoft SQL Server
Server	clouddw-server-da.database.windows.net
Authentication type	SQL Login
User name	clouddw-admin
Password	•••••••••

☑ Remember password

Click Connect. Azure Data Studio will ask to update firewall rules in order to access DW. We should specify the Azure account and authenticate it. It will ask to go to the URL https://microsoft.com/devicelogin and enter User code. As a result, we should get this message:

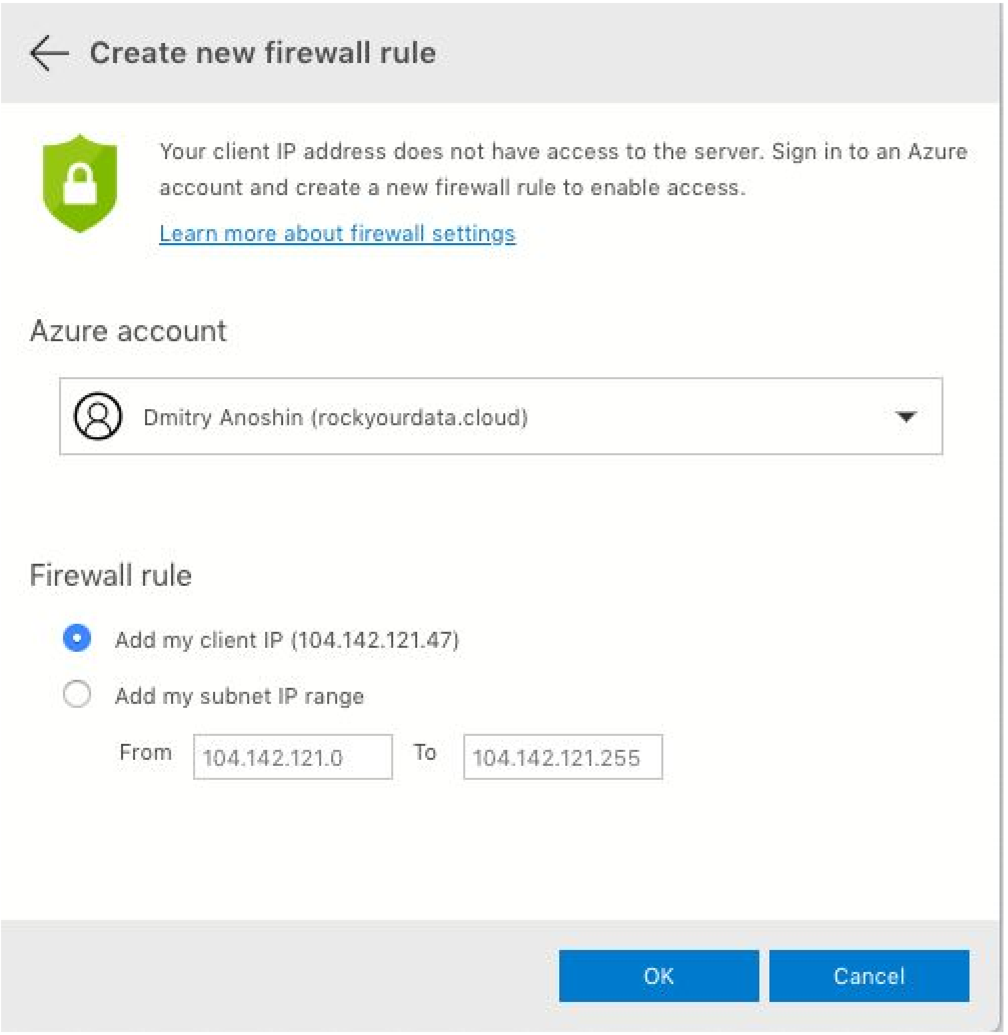


Microsoft

Azure Data Studio

You have signed in to the Azure Data Studio application on your device. You may now close this window.

Then we can update firewall rule by click Ok. It will connect us to the DW.



Create new firewall rule

Your client IP address does not have access to the server. Sign in to an Azure account and create a new firewall rule to enable access.

Learn more about firewall settings

Azure account

Dmitry Anoshin (rockyourdata.cloud)

Firewall rule

● Add my client IP (104.142.121.47)

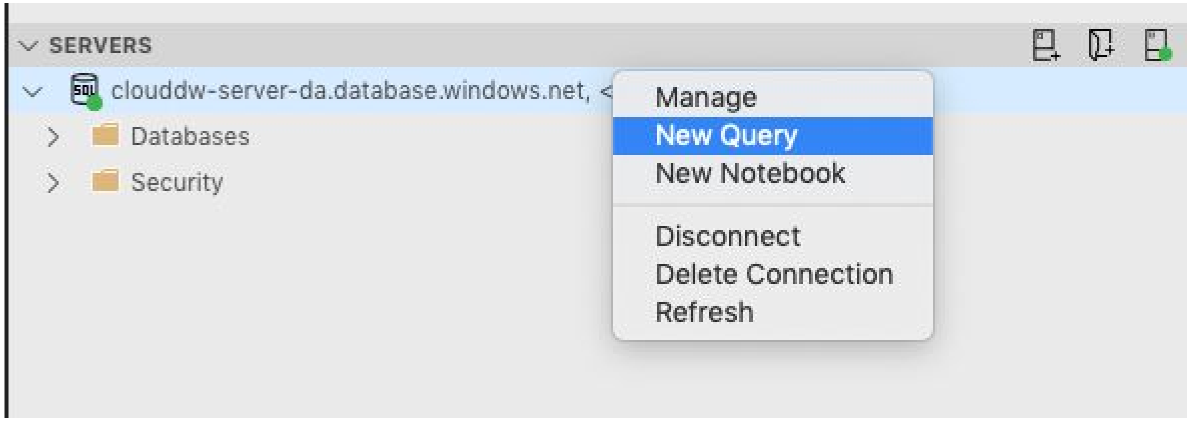
○ Add my subnet IP range

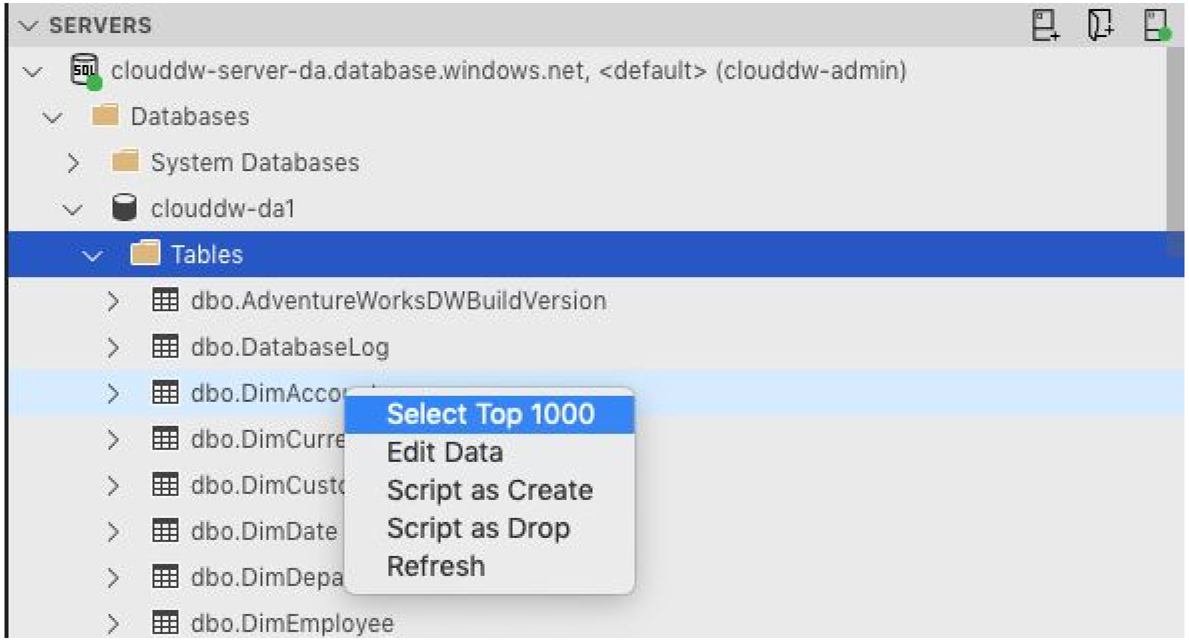
From 104.142.121.0 To 104.142.121.255

If we do everything right, we should connect Azure SQL DW.

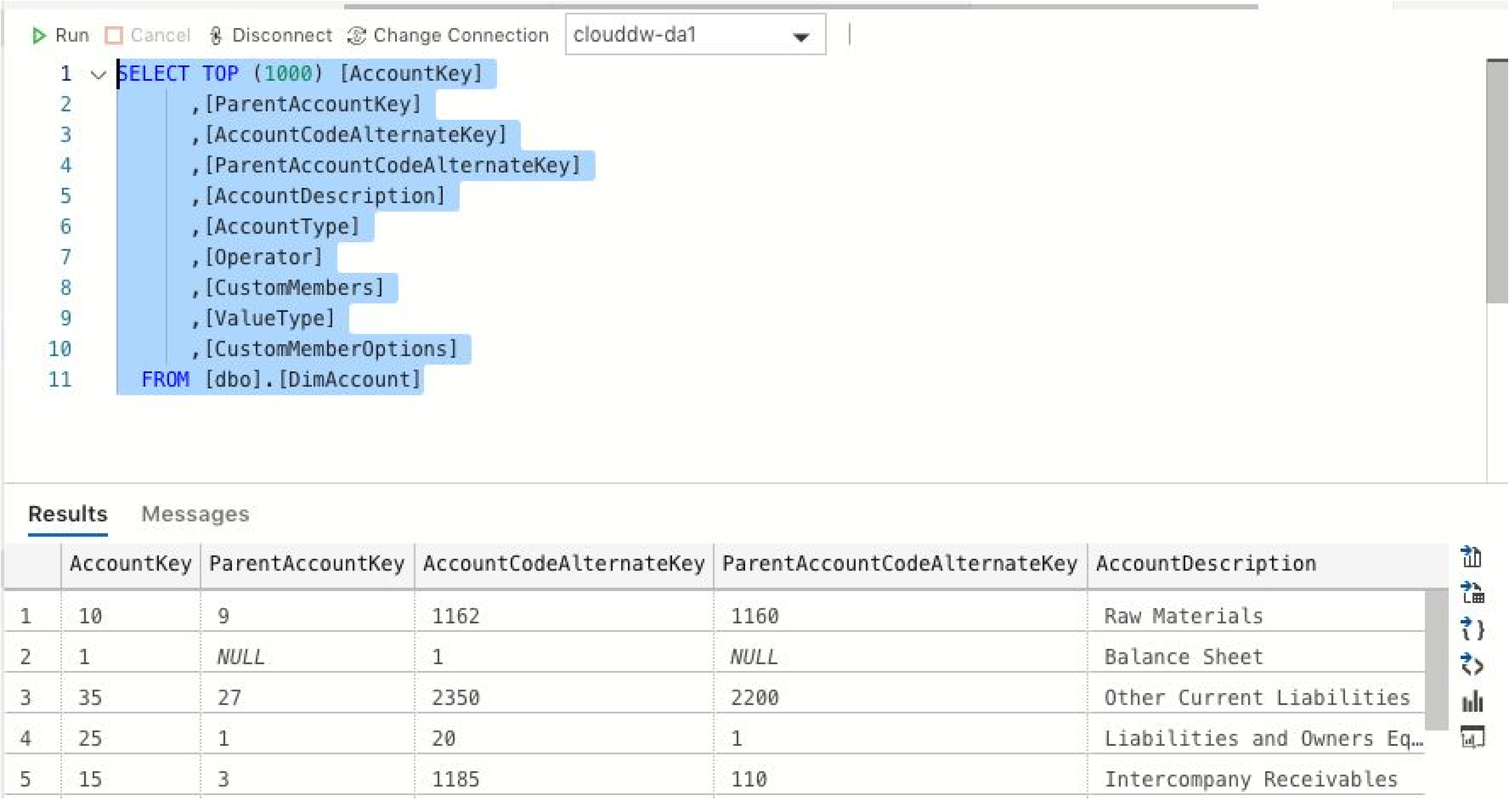
9. Let’s create a query. Click the right button on server name and choose New Query. It will open a new tab, where we can run a query.

Also, you can explore the tables and generate queries:





It will give us a sample output:

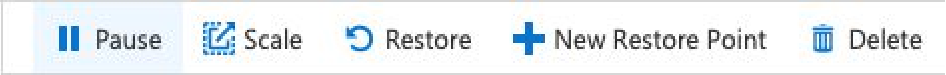


```
SELECT TOP (1000) [AccountKey]
      ,[ParentAccountKey]
      ,[AccountCodeAlternateKey]
      ,[ParentAccountCodeAlternateKey]
      ,[AccountDescription]
      ,[AccountType]
      ,[Operator]
      ,[CustomMembers]
      ,[ValueType]
      ,[CustomMemberOptions]
  FROM [dbo].[DimAccount]
```

	AccountKey	ParentAccountKey	AccountCodeAlternateKey	ParentAccountCodeAlternateKey	AccountDescription
1	10	9	1162	1160	Raw Materials
2	1	*NULL*	1	*NULL*	Balance Sheet
3	35	27	2350	2200	Other Current Liabilities
4	25	1	20	1	Liabilities and Owners Eq…
5	15	3	1185	110	Intercompany Receivables

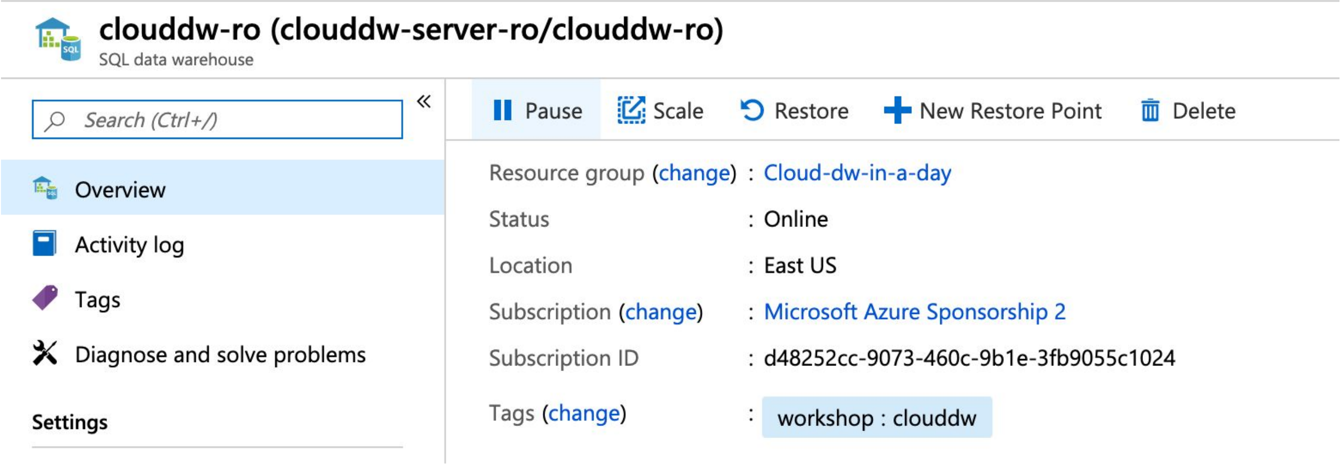
You may notice that the Azure Data Studio allows us to create charts and build dashboards. It could be useful for ad-hoc analysis.

10. Now we should pause DW using the Azure portal. Click **Pause** button on the top left of your DW overview page and confirm that you want to pause in pop-up window.



Exercise 3: Pause Azure SQL DW

1. We should pause our DW cluster. Go to Azure Portal -> All resources and find your instance of DW.
2. Click Pause:



clouddw-ro (clouddw-server-ro/clouddw-ro)

SQL data warehouse

Resource group (change) : Cloud-dw-in-a-day

Status : Online

Location : East US

Subscription (change) : Microsoft Azure Sponsorship 2

Subscription ID : d48252cc-9073-460c-9b1e-3fb9055c1024

Tags (change) : workshop : clouddw

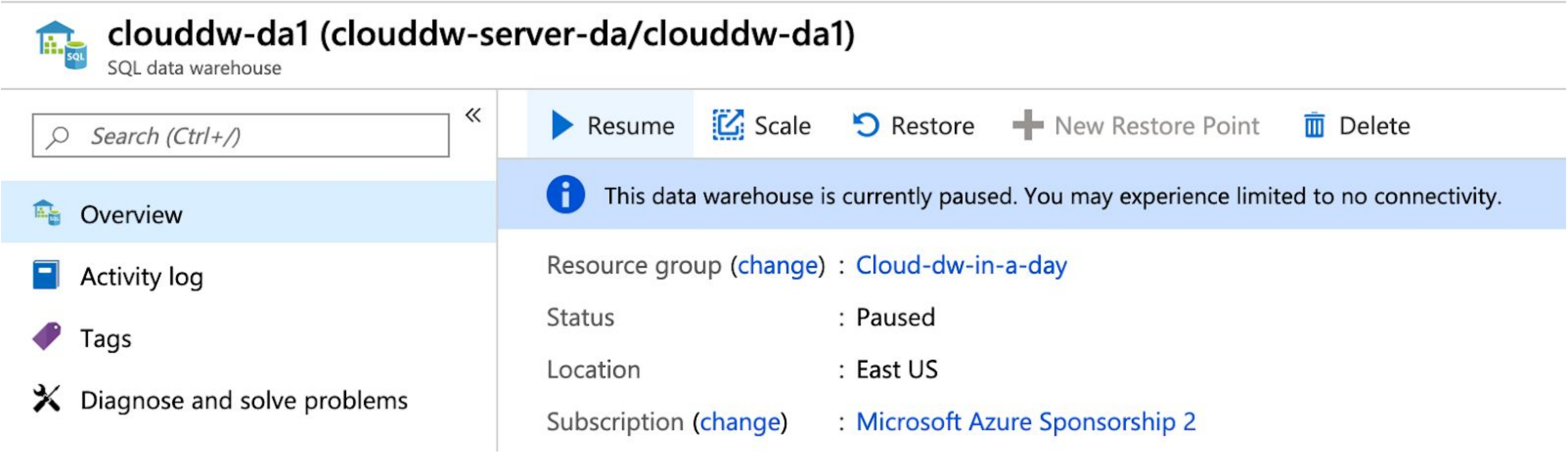
Lab 02: Design and Query Azure Data Warehouse

In this lab, we will continue with Azure SQL DW that we created in lab 01. We will apply the theory about table design. It includes:

- Distributed tables
- Partitions
- Indexes
- Statistics

Exercise 1: Resume SQL DW

1. We should resume our Azure SQL DW. Go to the Azure portal -> AllResourcess and choose your SQL DW.



clouddw-da1 (clouddw-server-da/clouddw-da1)

SQL data warehouse

This data warehouse is currently paused. You may experience limited to no connectivity.

Resource group (change) : Cloud-dw-in-a-day

Status : Paused

Location : East US

Subscription (change) : Microsoft Azure Sponsorship 2

Exercise 2: Create Tables with default options

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

1. Open SQL Client (Azure Data Studio or similar) and connect SQL DW.
2. We will create new tables and observe their default distribution methods and indexes. In query window run the following queries. You can copy them from Lab02.sql file.

Table without explicit distribution method	Table with the explicit distribution method
CREATE TABLE [dbo].[FactInternetSalesWithoutDistr] ([ProductKey] [int] NOT NULL, [OrderDateKey] [int] NOT NULL, [DueDateKey] [int] NOT NULL, [ShipDateKey] [int] NOT NULL, [CustomerKey] [int] NOT NULL, [PromotionKey] [int] NOT NULL, [CurrencyKey] [int] NOT NULL, [SalesTerritoryKey] [int] NOT NULL, [SalesOrderNumber] [nvarchar](20) NOT NULL, [SalesOrderLineNumber] [tinyint] NOT NULL, [RevisionNumber] [tinyint] NOT NULL, [OrderQuantity] [smallint] NOT NULL, [UnitPrice] [money] NOT NULL, [ExtendedAmount] [money] NOT NULL, [UnitPriceDiscountPct] [float] NOT NULL, [DiscountAmount] [float] NOT NULL,	CREATE TABLE [dbo].[FactInternetSalesWithDistr] ([ProductKey] [int] NOT NULL, [OrderDateKey] [int] NOT NULL, [DueDateKey] [int] NOT NULL, [ShipDateKey] [int] NOT NULL, [CustomerKey] [int] NOT NULL, [PromotionKey] [int] NOT NULL, [CurrencyKey] [int] NOT NULL, [SalesTerritoryKey] [int] NOT NULL, [SalesOrderNumber] [nvarchar](20) NOT NULL, [SalesOrderLineNumber] [tinyint] NOT NULL, [RevisionNumber] [tinyint] NOT NULL, [OrderQuantity] [smallint] NOT NULL, [UnitPrice] [money] NOT NULL, [ExtendedAmount] [money] NOT NULL, [UnitPriceDiscountPct] [float] NOT NULL, [DiscountAmount] [float] NOT NULL,

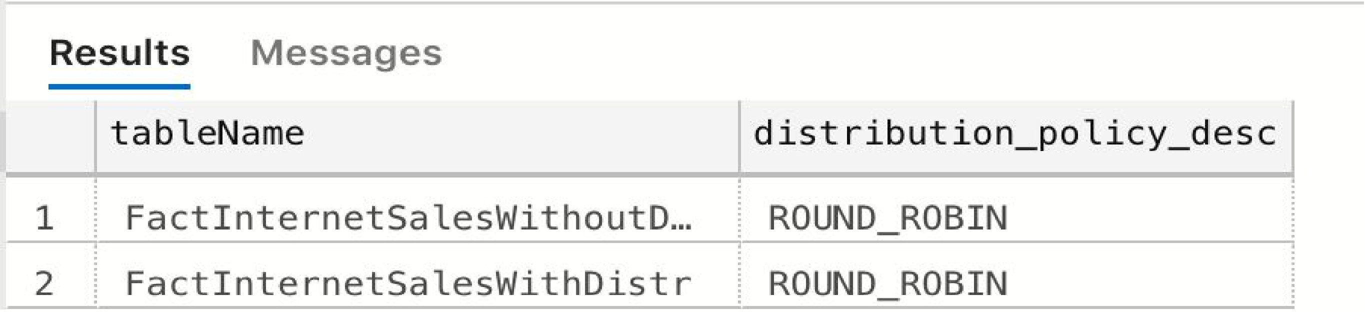
[ProductStandardCost] [money] NOT NULL, [TotalProductCost] [money] NOT NULL, [SalesAmount] [money] NOT NULL, [TaxAmt] [money] NOT NULL, [Freight] [money] NOT NULL, [CarrierTrackingNumber] [nvarchar](25) NULL, [CustomerPONumber] [nvarchar](25) NULL);	[ProductStandardCost] [money] NOT NULL, [TotalProductCost] [money] NOT NULL, [SalesAmount] [money] NOT NULL, [TaxAmt] [money] NOT NULL, [Freight] [money] NOT NULL, [CarrierTrackingNumber] [nvarchar](25) NULL, [CustomerPONumber] [nvarchar](25) NULL) WITH (DISTRIBUTION = ROUND_ROBIN, CLUSTERED COLUMNSTORE INDEX);

These queries create two new tables. If you don't see them refresh the server connection. In Azure Data Studio right click on the server name and select **Refresh**.

3. We can check their the distribution styles using system tables. We should run the following 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('FactInternetSalesWithoutDistr')
   OR ptdp.object_id = object_id('FactInternetSalesWithDistr')
   ```

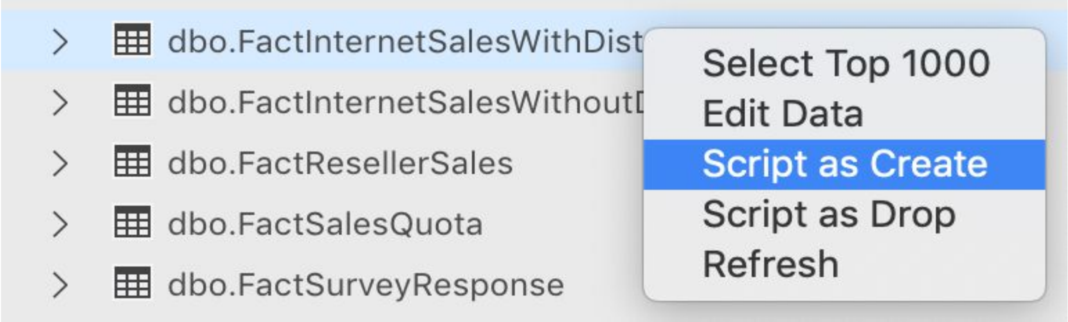
The result of the query shows that both tables have a distribution method of round-robin:



	tableName	distribution_policy_desc
1	FactInternetSalesWithoutD…	ROUND_ROBIN
2	FactInternetSalesWithDistr	ROUND_ROBIN

4. 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.

   We should check DDL of the table FactInternetSalesWithoutDistr and find that it is true. Click on table name with the right button in Azure Data Studio and choose **Script as Create**:



And we will see the COLUMNSTORE index:

```
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
GO
```

Exercise 3: Creating Distributed Tables

In this exercise, we 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. Let's create an Order table by running this query:

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

2. Then we will insert 60 rows into this table using this function:

```
SET NOCOUNT ON
DECLARE @i INT SET @i = 1
DECLARE @date DATETIME SET @date = dateadd(mi,@i,'2019-08-01') WHILE (@i <= 60)
BEGIN
INSERT INTO dbo.Orders (OrderDate) SELECT @date
SET @i = @i+1;
END;
```

We can check that we have data by running simple SQL query:

```
select count(*) from dbo.Orders;
```

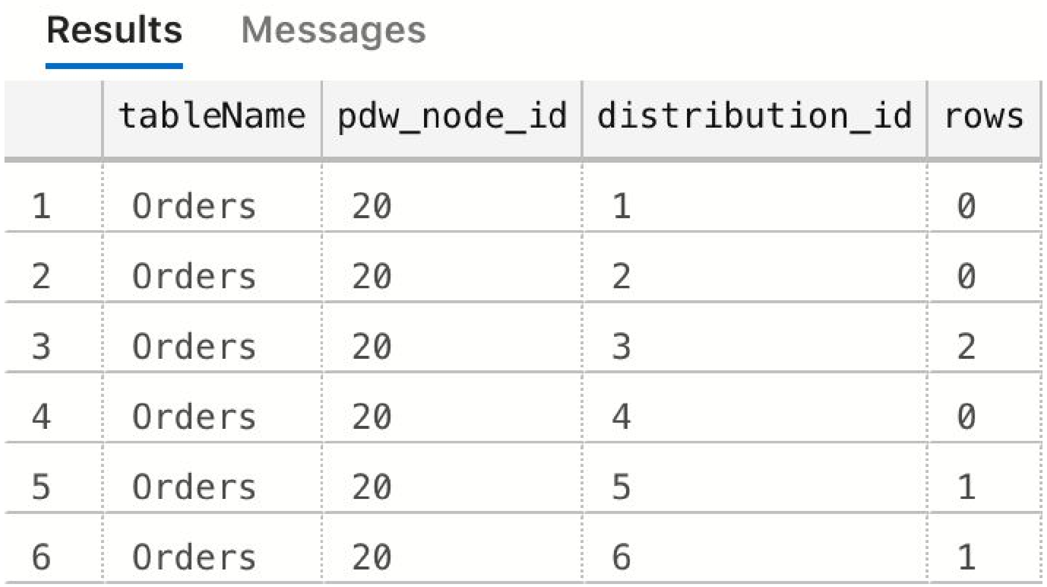
It should give us 60.

3. Using system views we can see how data was distributed across the distributions using the round robin distribution method. We should run this query:

```
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
```

A distribution is the basic unit of storage and processing for parallel queries that run on distributed data. When SQL Data Warehouse runs a query, the work is divided into 60 smaller queries that run in parallel. Each of the 60 smaller queries runs on one of the data distributions. Each Compute node manages one or more of the 60 distributions. A data warehouse with maximum compute resources has one distribution per Compute node. A data warehouse with minimum compute resources has all the distributions on one compute node.

In the results we can see that data was distributed somewhat evenly across the 60 distributions:



	tableName	pdw_node_id	distribution_id	rows
1	Orders	20	1	0
2	Orders	20	2	0
3	Orders	20	3	2
4	Orders	20	4	0
5	Orders	20	5	1
6	Orders	20	6	1

4. Next, we will see how hash is working.

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

Insert 60 rows using function below:

```
SET NOCOUNT ON
DECLARE @i INT SET @i = 1
DECLARE @date DATETIME SET @date = dateadd(mi,@i,'2019-08-01')
WHILE (@i <= 60)
BEGIN
INSERT INTO dbo.Orders2 (OrderDate) SELECT @date
SET @i = @i+1;
END
```

Check the number of rows:

```
select count(*) from dbo.Orders2;
```

View distributions for Orders2 table:

```
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')
```

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 DW hash function is deterministic (each row is assigned to one distribution based on the hash).
A hash distributed table can deliver the highest query performance for joins and aggregations on large tables.

> *To shard data into a hash-distributed table, SQL Data Warehouse uses a hash function to deterministically assign each row to one distribution. In the table definition, one of the columns is designated as the distribution column. The hash function uses the values in the distribution column to assign each row to a distribution.*

Let’s reload data with more distinct values for our hash key. First we should truncate table:

```
TRUNCATE TABLE dbo.Orders2;
```

Then run new function

```
SET NOCOUNT ON
DECLARE @i INT SET @i = 1
DECLARE @date DATETIME SET @date = dateadd(mi,@i,'2019-08-01')
WHILE (@i <= 10)
BEGIN
INSERT INTO dbo.Orders2 (OrderDate) SELECT @date
INSERT INTO dbo.Orders2 (OrderDate) SELECT dateadd(week,1,@date)
INSERT INTO dbo.Orders2 (OrderDate) SELECT dateadd(week,2,@date)
INSERT INTO dbo.Orders2 (OrderDate) SELECT dateadd(week,3,@date)
INSERT INTO dbo.Orders2 (OrderDate) SELECT dateadd(week,4,@date)
INSERT INTO dbo.Orders2 (OrderDate) SELECT dateadd(week,5,@date)
SET @i = @i+1;
END
```

We can see our distributions

```
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')
```

Now, we can see that our data is better distributed. Best practices recommend using the hash distribution as much as possible to minimize data movement, which will improve query performance. Especially, for hash, it is good practice to pick-up the column that has at least 60 rows.

Exercise 4: Replicating Tables

In this exercise, we 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. Create two new table with distribution style Round-robin using CREATE AS SELECT statement:

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

   And

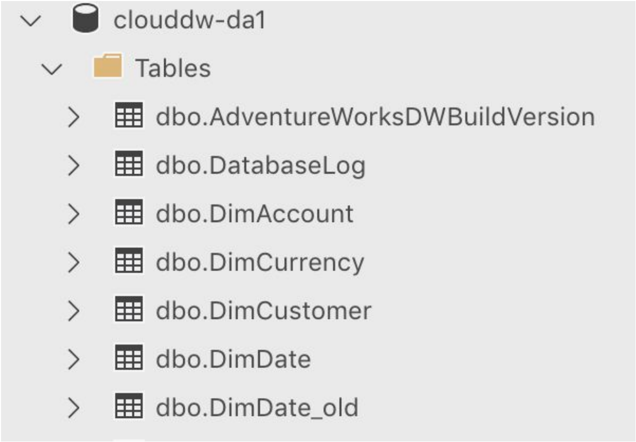
```
CREATE TABLE dbo.DimDate_REPLICATE
WITH
(
CLUSTERED COLUMNSTORE INDEX,
DISTRIBUTION = ROUND_ROBIN
)
AS SELECT * FROM dbo.DimDate
OPTION (LABEL = 'CTAS : DimDate_REPLICATE')
```

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.

2. We will switch the tables names:

```
RENAME OBJECT dbo.DimDate to DimDate_old;
RENAME OBJECT dbo.DimDate_REPLICATE TO DimDate;
RENAME OBJECT dbo.DimSalesTerritory to DimSalesTerritory_old;
RENAME OBJECT dbo.DimSalesTerritory_REPLICATE TO DimSalesTerritory;
```

3. Refresh the connection. We should see our tables in Explorer. For example, on screen below we can see DimDate tables:

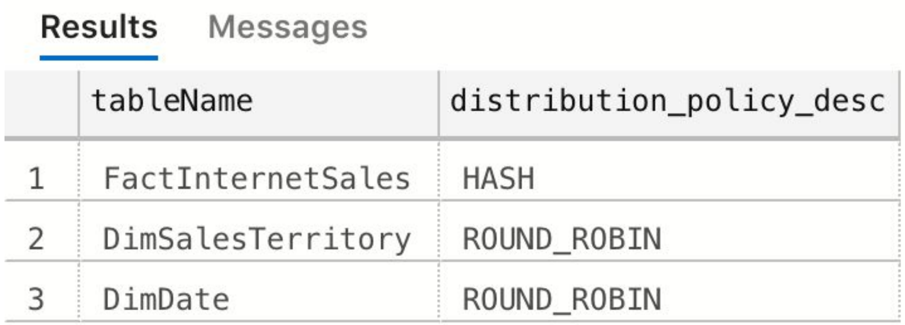


clouddw-da1
- Tables
  - dbo.AdventureWorksDWBuildVersion
  - dbo.DatabaseLog
  - dbo.DimAccount
  - dbo.DimCurrency
  - dbo.DimCustomer
  - dbo.DimDate
  - dbo.DimDate_old

Moreover, we can review the distribution style of new tables using system tables by running this 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 o.name in ('DimDate','DimSalesTerritory','FactInternetSales')
```

We should get this output:



	tableName	distribution_policy_desc
1	FactInternetSales	HASH
2	DimSalesTerritory	ROUND_ROBIN
3	DimDate	ROUND_ROBIN

We see that Dimension tables have Round Robin style and Fact table has Hash style.

4. Let's run the SQL and check the the distributions across DimDate table with Round Robin:

```
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 ('DimDate') ORDER BY distribution_id
```

Also, you can compare the difference with DimDate_old (HASH distribution) by running the following function:

```
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 ('DimDate_old') ORDER BY distribution_id
```

5. Now, we can run SQL against our SQL DW. Execute the following query:

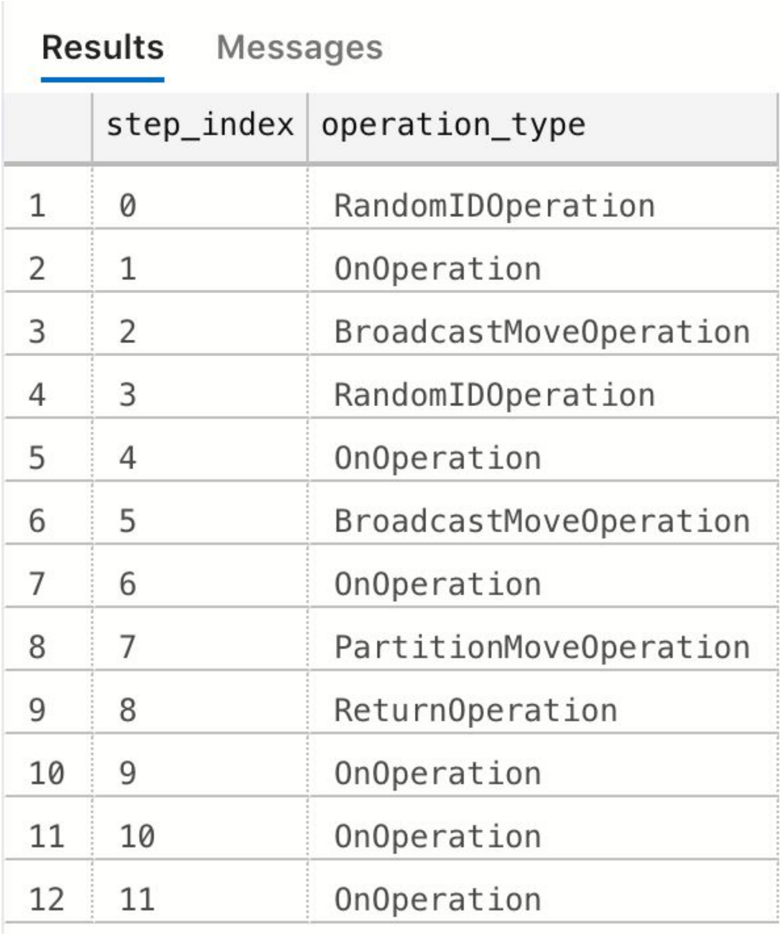
```
SELECT TotalSalesAmount = SUM(SalesAmount)
FROM dbo.FactInternetSales s
INNER JOIN dbo.DimDate d
```

```
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');
```

This query returns Total Sales Amount for 2004 year in North America. Moverover, we used the OPTION LABEL. It will help us to tracking down problem queries. For example, we can use the system tables in order to get information about all requests currently or recently active in SQL DW and all steps that make up a given request to query. The table above helped us to find our query. We will get the list of steps and operation types running the following 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';
```

This would result in the table:



	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	ReturnOperation
10	9	OnOperation
11	10	OnOperation
12	11	OnOperation

We have multiple Move operations such as BroadcastMoveOperation that is very expensive for queries. Both tables DimDate and DimSalesTerritory have Round Robin distribution method, there was a join of copies of each table in full to each Compute node.

6. Let’s try to improve performance for this query and change distribution styles for dimension tables:

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

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.DimSalesTerritory to DimSalesTerritory_RR;
RENAME OBJECT dbo.DimSalesTerritory_REPLICATE TO DimSalesTerritory;
RENAME OBJECT dbo.DimDate to DimDate_RR;
RENAME OBJECT dbo.DimDate_REPLICATE TO DimDate;
```

Now, we can execute our SQL query again that will help us to find Sales Amount in 2004 for NA:

```
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');
```

When query is finished, we should check the list of steps and operation types again using the same 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';
```

It will still have 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. That’s why we have to run our SQL one more time in order to get real list of steps:

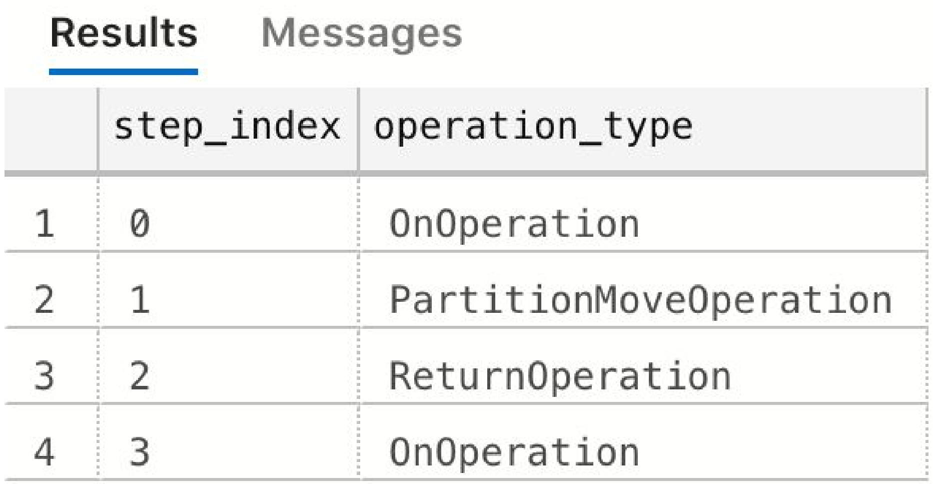
```
SELECT TotalSalesAmount = SUM(SalesAmount)
FROM dbo.FactInternetSales s
INNER JOIN dbo.DimDate d
ON d.DateKey = s.OrderDateKey
INNER JOIN dbo.DimSalesTerritory t
ON t.SalesTerritoryKey = s.SalesTerritoryKey
WHERE d.FiscalYear = 2004
AND t.SalesTerritoryGroup = 'North America'
OPTION (LABEL = 'STATEMENT:ReplicatedTableQuery_lucky');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

Then quickly check the operation types:

```
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_lucky';
```

It will be a huge improvement. We don't have anymore the BroadcastMoveOperation:



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

This is how we can tune our queries and make more efficient design of Azure SQL DW.

Exercise 5: Managing Statistics

In this exercise, we 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.

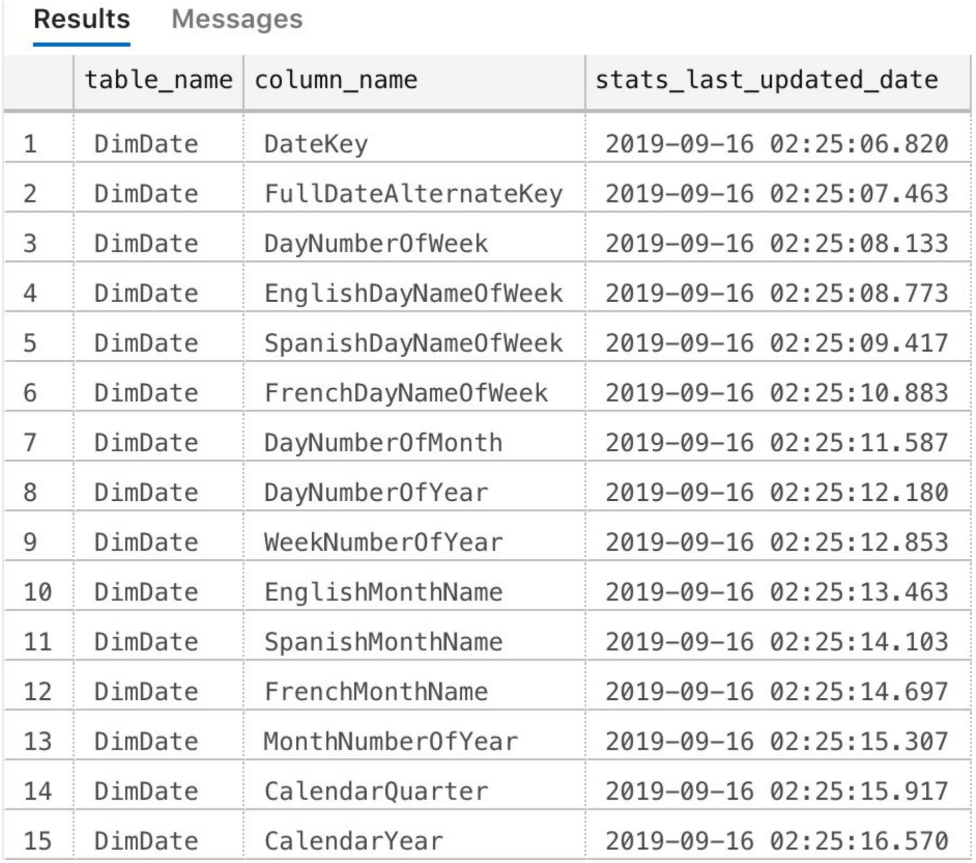
1. We will check the statistics status for our DImension tables:

```
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');
```

There won't be anything until we collect the statistics for columns:

```
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);
```

If we run query from step 1 again, we will see the updated statistics:



	table_name	column_name	stats_last_updated_date
1	DimDate	DateKey	2019-09-16 02:25:06.820
2	DimDate	FullDateAlternateKey	2019-09-16 02:25:07.463
3	DimDate	DayNumberOfWeek	2019-09-16 02:25:08.133
4	DimDate	EnglishDayNameOfWeek	2019-09-16 02:25:08.773
5	DimDate	SpanishDayNameOfWeek	2019-09-16 02:25:09.417
6	DimDate	FrenchDayNameOfWeek	2019-09-16 02:25:10.883
7	DimDate	DayNumberOfMonth	2019-09-16 02:25:11.587
8	DimDate	DayNumberOfYear	2019-09-16 02:25:12.180
9	DimDate	WeekNumberOfYear	2019-09-16 02:25:12.853
10	DimDate	EnglishMonthName	2019-09-16 02:25:13.463
11	DimDate	SpanishMonthName	2019-09-16 02:25:14.103
12	DimDate	FrenchMonthName	2019-09-16 02:25:14.697
13	DimDate	MonthNumberOfYear	2019-09-16 02:25:15.307
14	DimDate	CalendarQuarter	2019-09-16 02:25:15.917
15	DimDate	CalendarYear	2019-09-16 02:25:16.570

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

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

The following SQL statement updates the statistics for all the columns in the DimDate table:

```
UPDATE STATISTICS dbo.DimDate;
```

Exercise 6: Adding Partitions

In this exercise, we 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. We will start from creating new OrderPartition table and will specify partitions in DDL. Execute the following DDL command:

```
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'
)
)
);
```

Then we will generate dummy data and insert into the table:

```
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
```

You can check the number of rows, should be 60:

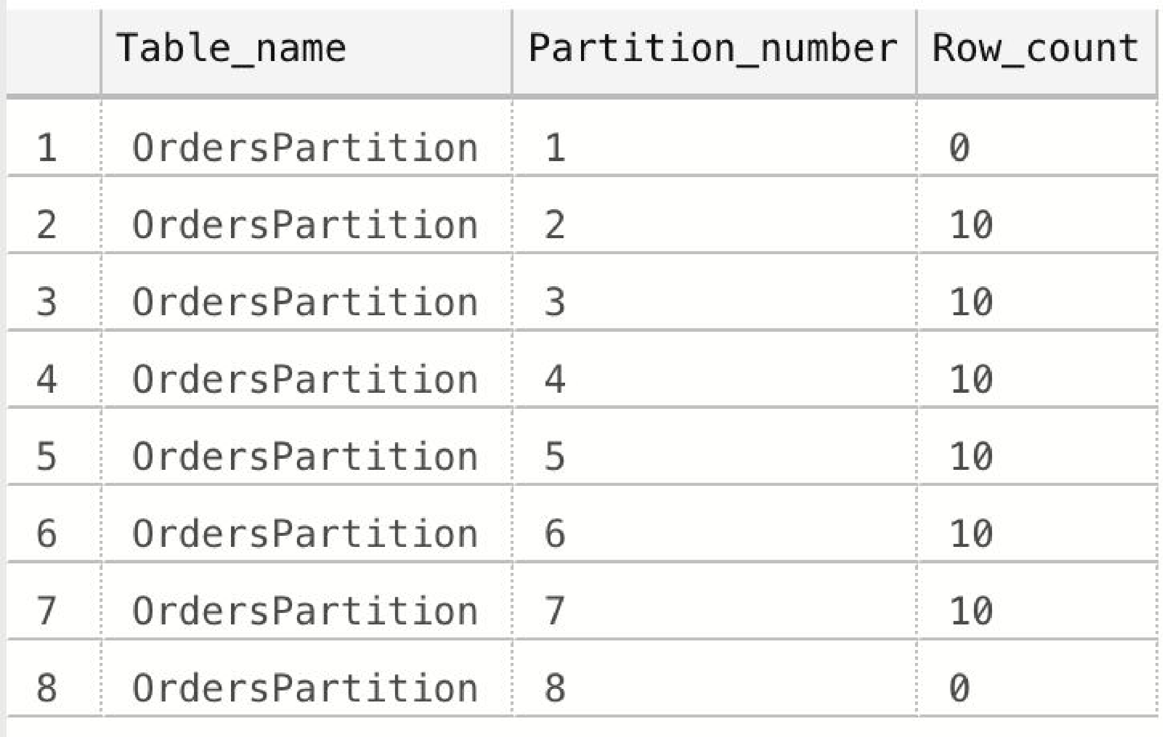
```
SELECT COUNT(*) FROM OrdersPartition;
```

2. Using system tables, we can find information about partitions for particular table:

```
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;
```

We will get this output:



	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

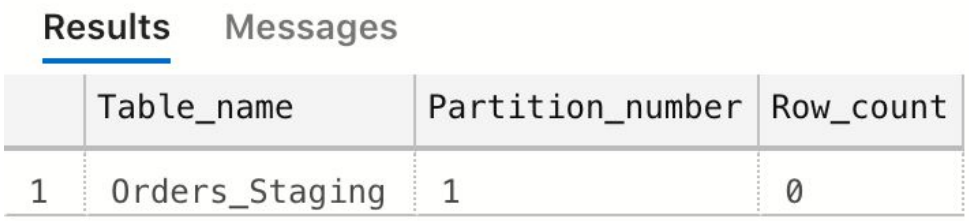
3. Let’s create another table, without partitions:

```
CREATE TABLE dbo.Orders_Staging
(OrderID int IDENTITY(1,1) NOT NULL
,OrderDate datetime NOT NULL
,OrderDescription char(15) DEFAULT 'NewOrder'
);
```

Using the query from step 2, we can check the partitions:

```
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;
```

There is no partitions in this table:



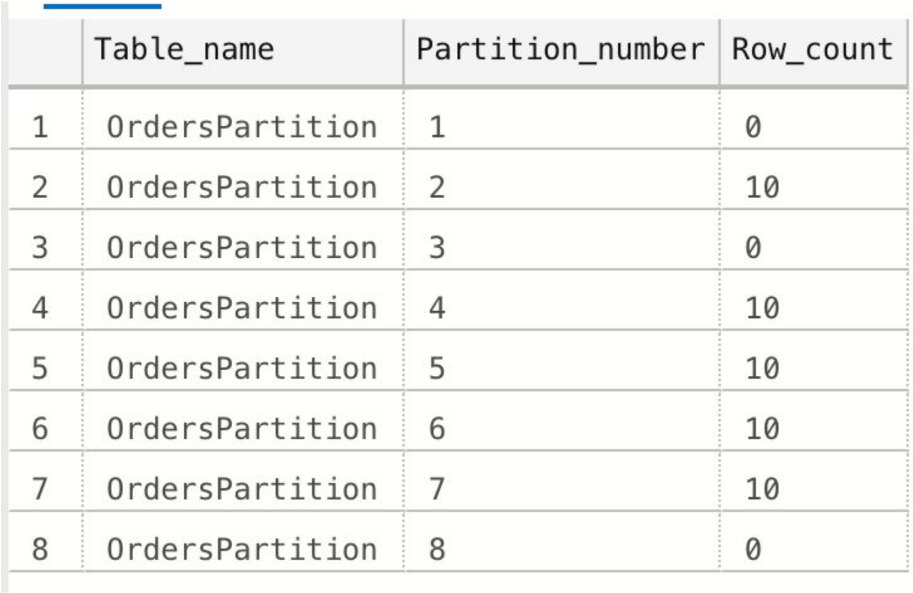
	Table_name	Partition_number	Row_count
1	Orders_Staging	1	0

4. We can use the ALTER TABLE command in order to switch out the data of partition 3 of the OrdersPartition table into the default partition of the Orders_Staging table.

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

After this step, we will review the partition information for the OrdersPartition table:

```sql
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;
```

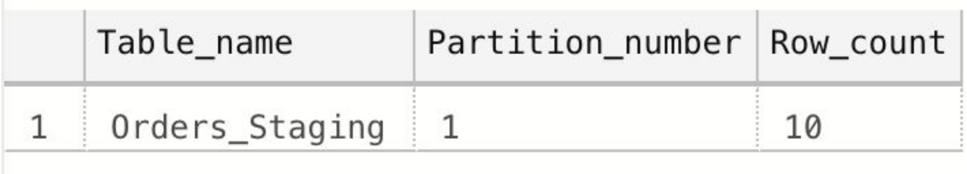


	Table_name	Partition_number	Row_count
1	OrdersPartition	1	0
2	OrdersPartition	2	10
3	OrdersPartition	3	0
4	OrdersPartition	4	10
5	OrdersPartition	5	10
6	OrdersPartition	6	10
7	OrdersPartition	7	10
8	OrdersPartition	8	0

And for the Orders_Staging table:

```
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;
```

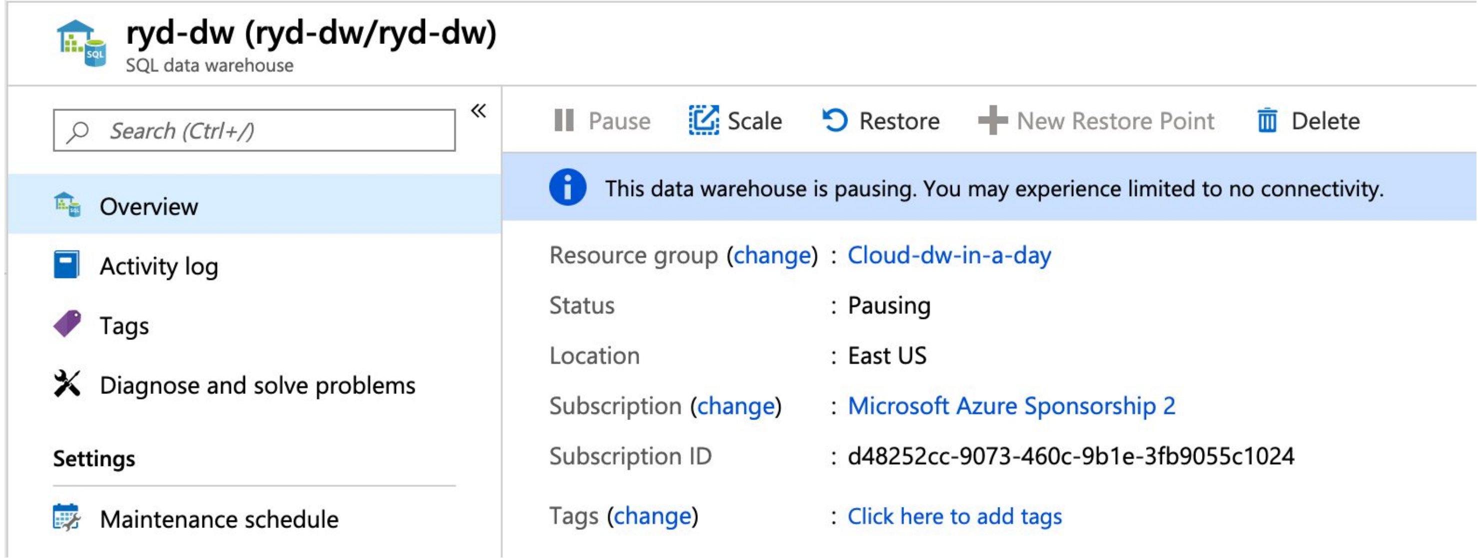
So, we physically moved rows from one table to another and the scope was defined by Partition.



	Table_name	Partition_number	Row_count
1	Orders_Staging	1	10

Exercise 7: Pause Azure SQL DW

We should pause our cluster of Azure SQL DW. Go to Azure portal, find your resource and click Pause.



This data warehouse is pausing. You may experience limited to no connectivity.

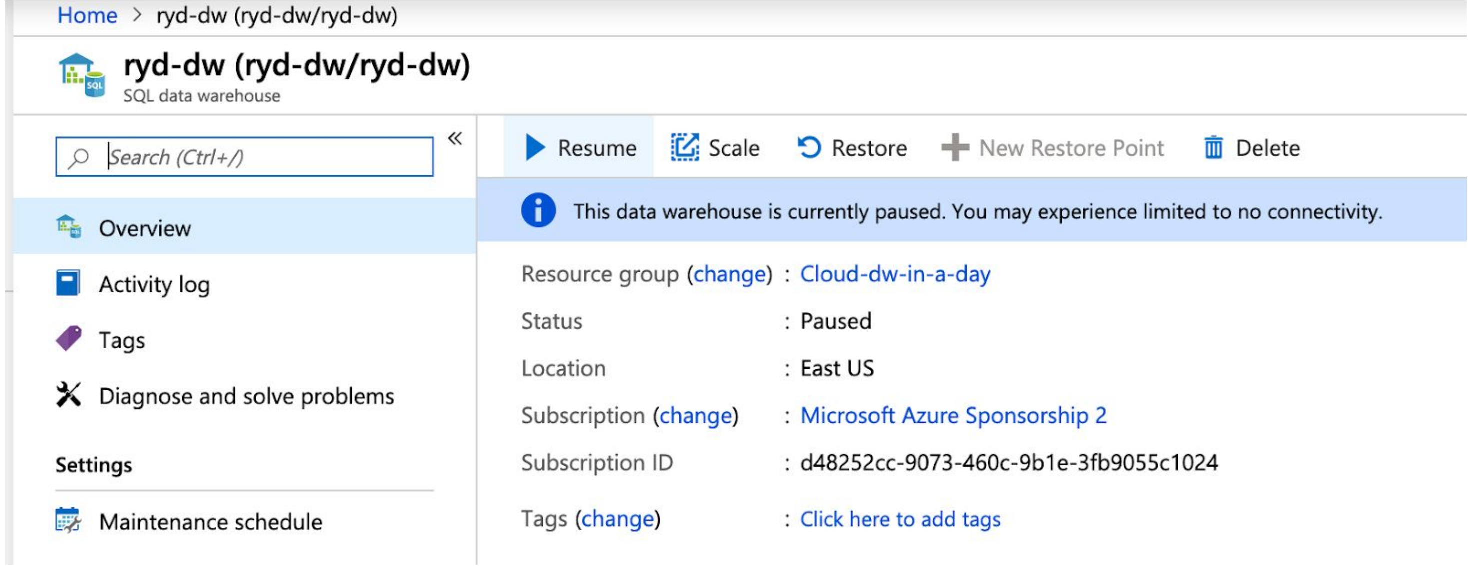
Resource group (change)	: Cloud-dw-in-a-day
Status	: Pausing
Location	: East US
Subscription (change)	: Microsoft Azure Sponsorship 2
Subscription ID	: d48252cc-9073-460c-9b1e-3fb9055c1024
Tags (change)	: Click here to add tags

Lab3: Loading Data into Azure SQL DW

In this lab, you will load data and move data into your Azure SQL Data Warehouse, using Azure Blob Storage, Azure Data Factory, Azure Stream Analytics, and Polybase.

Exercise 1: Resume Azure SQL DW

Go to the Azure Portal, find your SQL DW in Azure resources and click Start.



This data warehouse is currently paused. You may experience limited to no connectivity.

Resource group (change) : Cloud-dw-in-a-day

Status : Paused

Location : East US

Subscription (change) : Microsoft Azure Sponsorship 2

Subscription ID : d48252cc-9073-460c-9b1e-3fb9055c1024

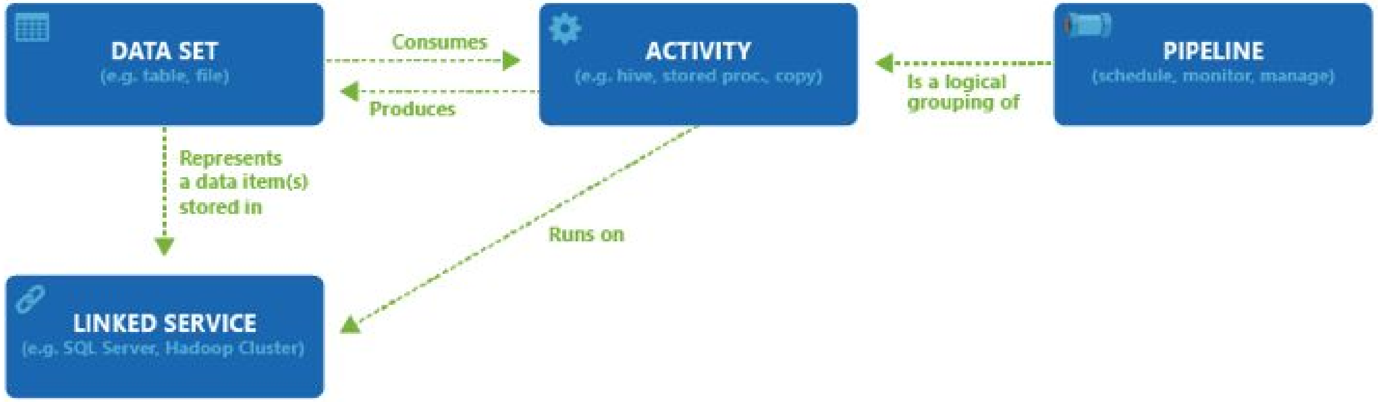
Tags (change) : Click here to add tags

Exercise 2: Load data with Azure Data Factory

In this exercise, we will first create a table in the Azure SQL Data Warehouse sample database, then create an Azure Data Factory to integrate and move data from Azure blob storage to the table in Azure SQL Data Warehouse. We will use sample txt file that has sales data and product information. We will pick up this file from BLOB and load into DW using ADF.

The key terms for ADF:

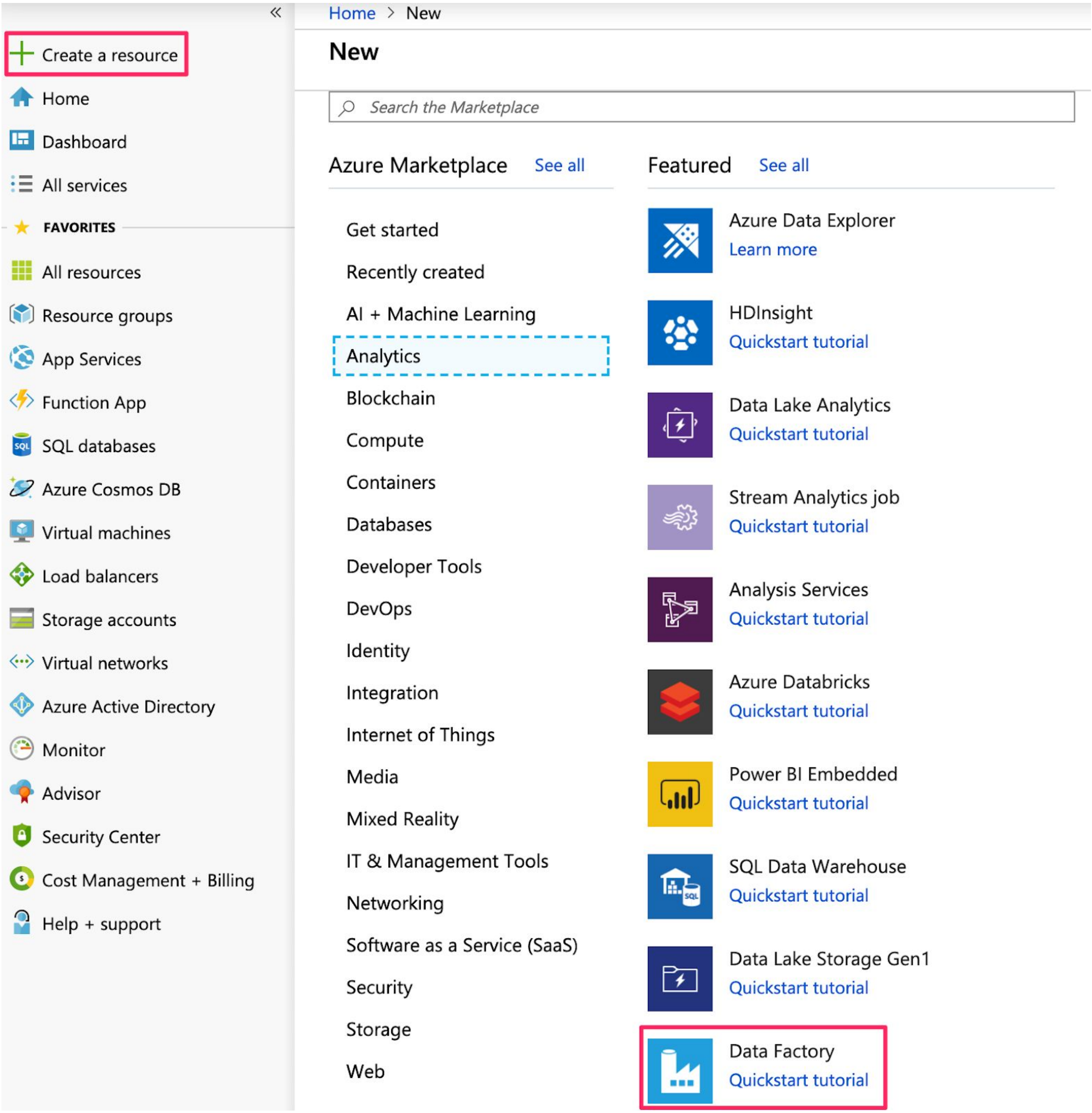
- **Activity -** define actions to perform on your data
- **Pipeline -** is a logical grouping of activities that together perform a task.
- **Dataset -** is a named view of data that simply points or references the data you want to use in your activities as inputs and outputs. Datasets identify data within different data stores, such as tables, files, folders, and documents.
- **Linked Services -** are much like connection strings, which define the connection information needed for Data Factory to connect to external resources. Think of it this way; the dataset represents the structure of the data within the linked data stores, and the linked service defines the connection to the data source.


DATA SET
(e.g. table, file)
Consumes
Produces
ACTIVITY
(e.g. hive, stored proc., copy)
Is a logical grouping of
PIPELINE
(schedule, monitor, manage)
Represents a data item(s) stored in
Runs on
LINKED SERVICE
(e.g. SQL Server, Hadoop Cluster)

1. Using Azure Data Studio make sure that your connection to DW is active. Then we will create new table - SuperStoreOrders using DDL below:

```
CREATE TABLE [dbo].[SuperStoreOrders]
(
    RowID [int] NOT NULL,
    OrderID [nvarchar](200) NOT NULL,
    OrderDate DATE NOT NULL,
    ShipDate DATE NOT NULL,
    ShipMode [nvarchar](200) NOT NULL,
    CustomerID [nvarchar](200) NOT NULL,
    CustomerName [nvarchar](200) NOT NULL,
    Segment [nvarchar](200) NOT NULL,
    Country [nvarchar](200) NOT NULL,
    City [nvarchar](200) NOT NULL,
    State [nvarchar](200) NOT NULL,
    PostalCode int,
    Region [nvarchar](200) NOT NULL,
    ProductID [nvarchar](200) NOT NULL,
    Category [nvarchar](200) NOT NULL,
    SubCategory [nvarchar](200) NOT NULL,
    ProductName [nvarchar](200) NOT NULL,
    Sales [money] NOT NULL,
    Quantity int,
    Discount [money] NOT NULL,
    Profit [money] NOT NULL
)
WITH
(
    DISTRIBUTION = HASH ( RowID ),
    CLUSTERED COLUMNSTORE INDEX
);
```

2. Now, we will move to Azure Data Factory. Click **Create Resource** (top left corner) in Azure Portal and choose **Analytics** -> **Data Factory**.



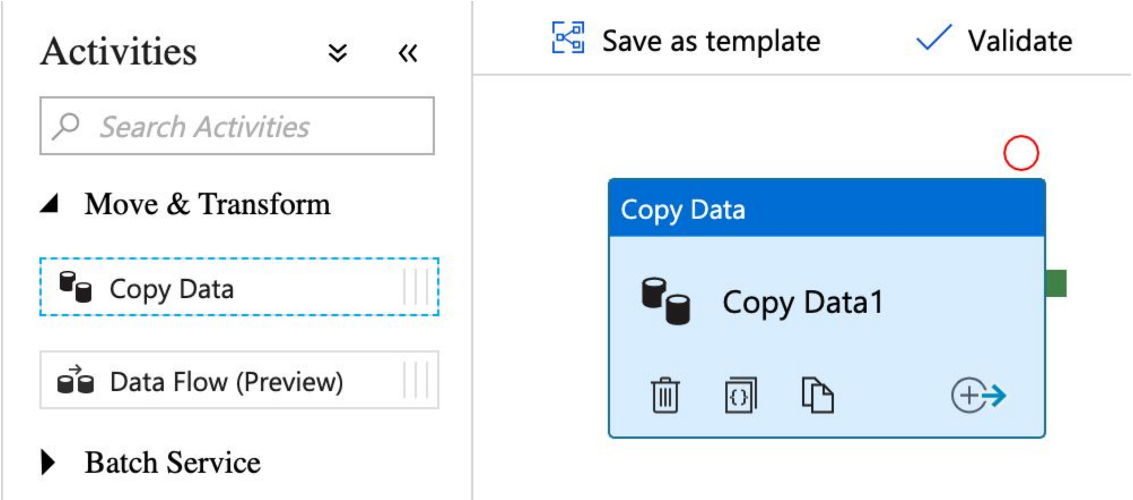
It will open a new tab with options. There is a table below with the options:

Name	clouddw-adf-<your initials>
Subscription	Microsoft Azure Sponsorship 2
Resource Group (use existing)	Cloud-dw-in-a-day
Version	V2
Location	East US

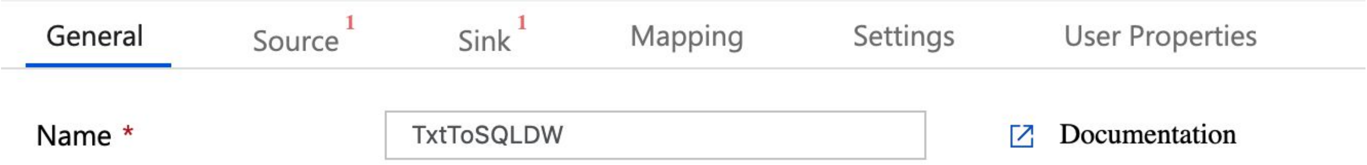
We don’t need GIT integration, deselect the checkbox. Then click **Create**. It will provision Azure Data Factory V2.

3. Navigate to Data Factory. (Notification pane on the right top will have a short-cut “go to resource”)Click on Author&Monitor. It will open a new tab with ADF information where we can create activities and learn more about ADF. We will create new Pipeline. Click on **Create Pipeline**.

Using Activities Pane, we should search for **Copy** activity and add this to the designer surface.

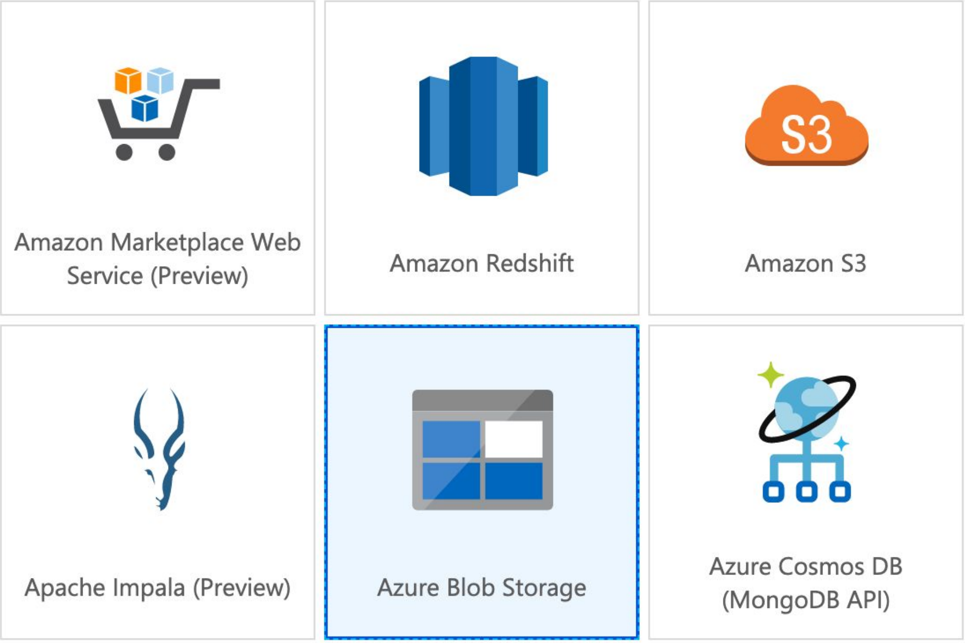


4. Currently, the Copy activity displays a Red exclamation point, signifying that the activity has not been configured. Next, we will fill the properties of Copy activity. In **General** tab add the name - **TxtToSQLDW**:

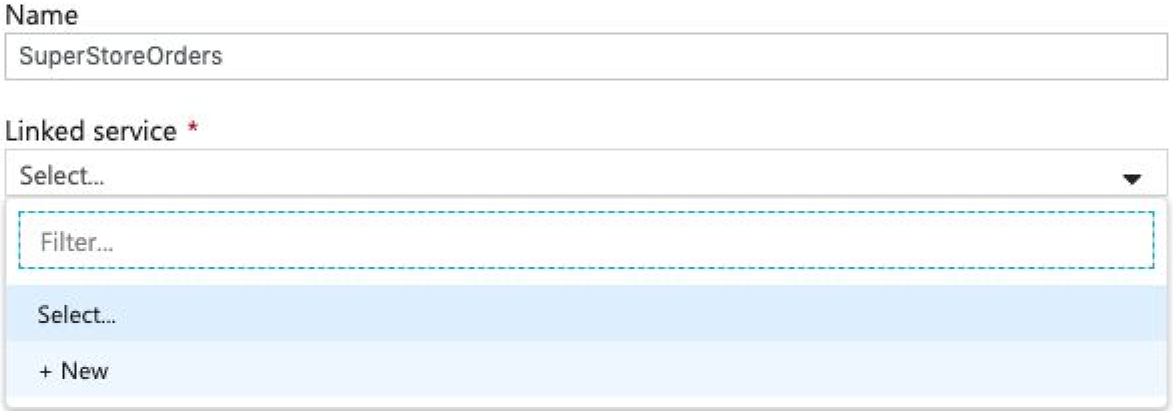


Name * TxtToSQLDW

Then we should move to the **Source** tab. Click **+New** and choose Azure Blob Storage.


Amazon Marketplace Web Service (Preview)
Amazon Redshift
Amazon S3
Apache Impala (Preview)
Azure Blob Storage
Azure Cosmos DB (MongoDB API)

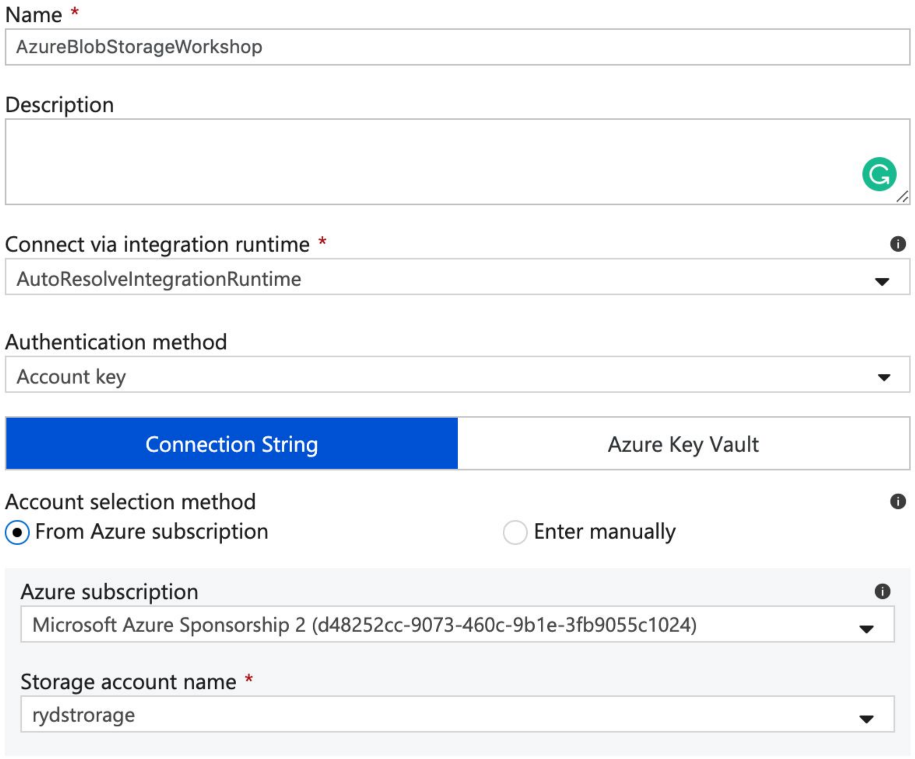
Click Continue and choose format type **DelimetedText** and click continue. We can give a name **SuperStoreOrders** and create new Linked Service:



Name
SuperStoreOrders

Linked service *
Select...

The options are similar for all:



Name *
AzureBlobStorageWorkshop

Description

Connect via integration runtime *
AutoResolveIntegrationRuntime

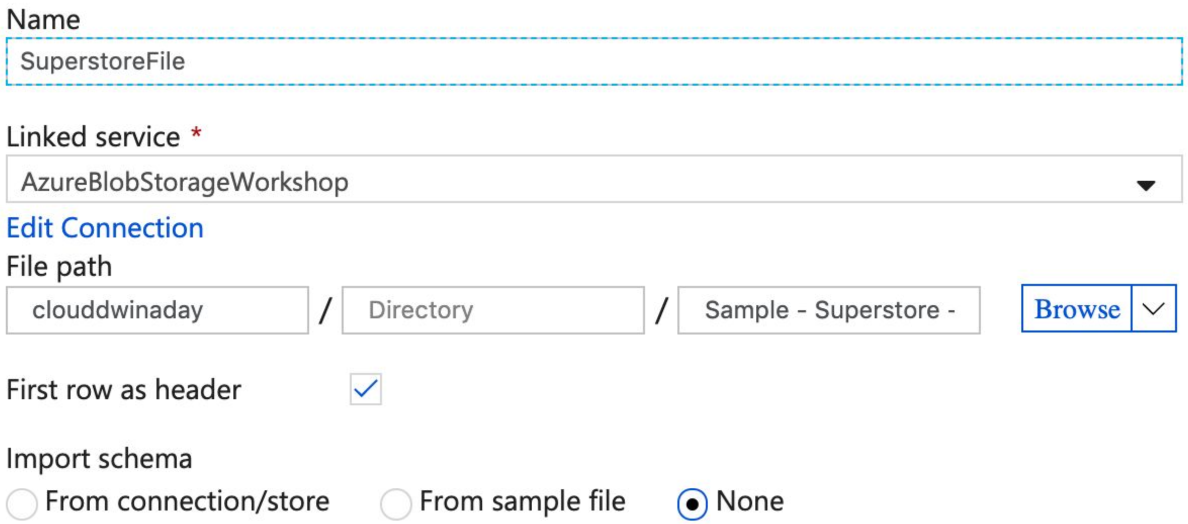
Authentication method
Account key

Account selection method
● From Azure subscription ○ Enter manually

Azure subscription
Microsoft Azure Sponsorship 2 (d48252cc-9073-460c-9b1e-3fb9055c1024)

Storage account name *
rydstrorage

Then we can set properties. Click **Finish** to see the next window. We need to choose file from Azure Blob and specify that we have first row as header. We don't want to import schema. But source fields should match target fields. Select file path using **Browse.** Click **Finish** when done.



Name: SuperstoreFile

Linked service *: AzureBlobStorageWorkshop

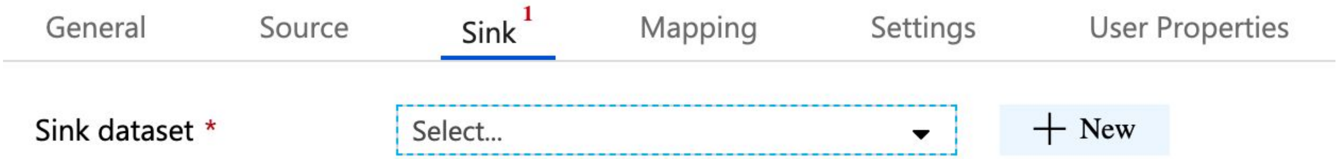
Edit Connection

File path: clouddwinaday / Directory / Sample - Superstore -

First row as header ☑

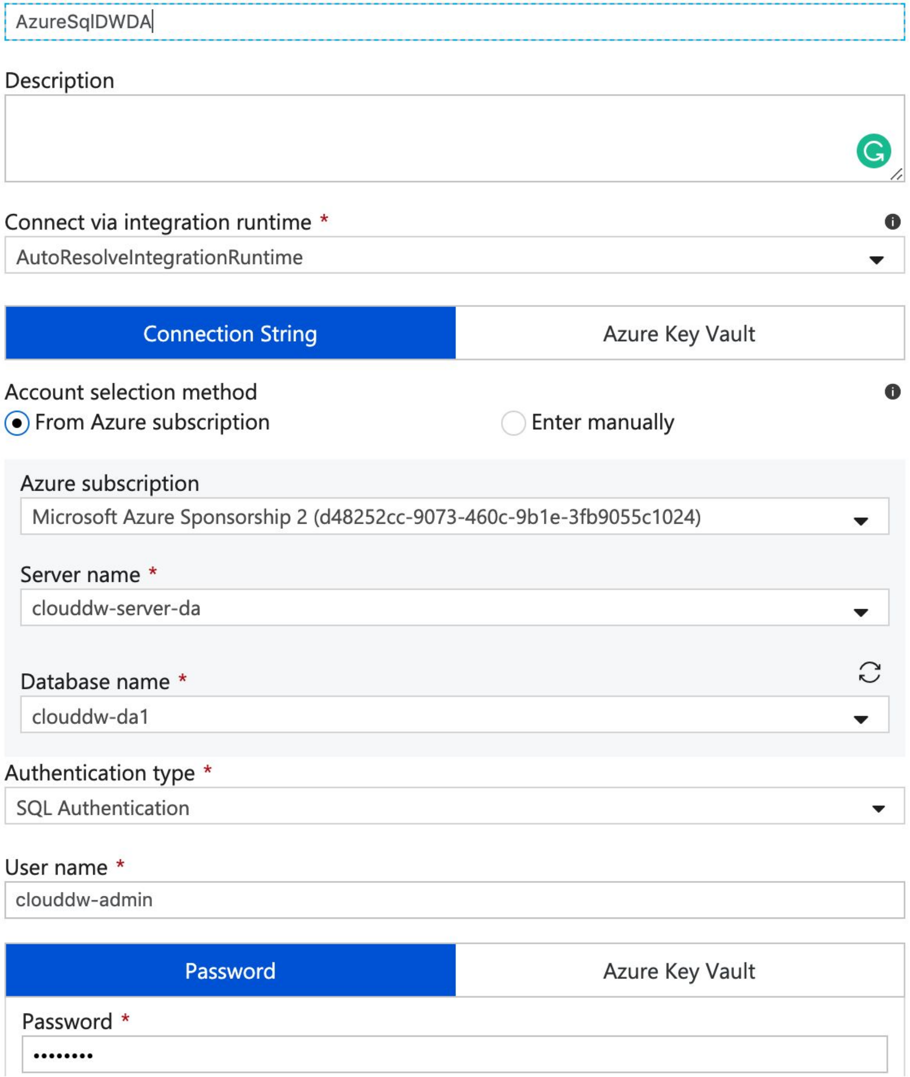
Import schema: ○ From connection/store ○ From sample file ● None

5. Move to the next tab - **Sink**. It is our target. We want to load data into Azure SQL DW. Click +New.



Sink dataset *: Select...

Choose Azure SQL DW icon and create new Linked service. The name is AzureSQLDW<your initials>:



AzureSqlDWDA

Description

Connect via integration runtime *

AutoResolveIntegrationRuntime

Account selection method

● From Azure subscription ○ Enter manually

Azure subscription

Microsoft Azure Sponsorship 2 (d48252cc-9073-460c-9b1e-3fb9055c1024)

Server name *

clouddw-server-da

Database name *

clouddw-da1

Authentication type *

SQL Authentication

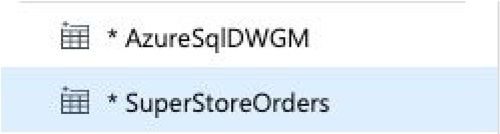
User name *

clouddw-admin

Password *

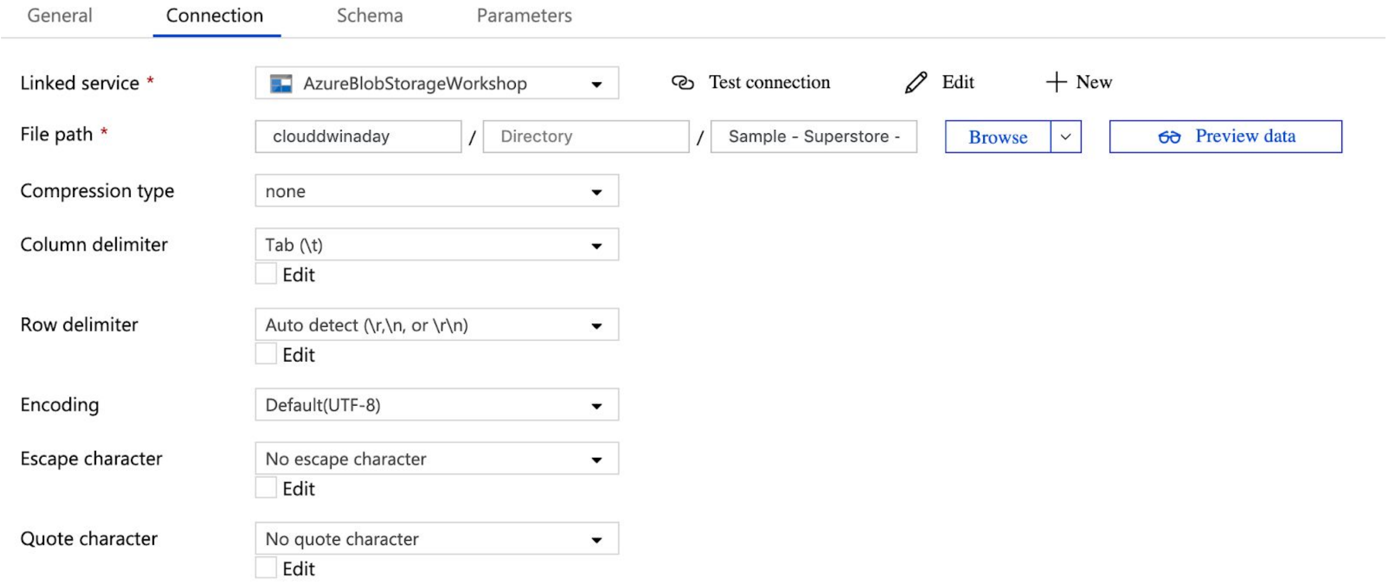
••••••••

Click **Test connection**, make sure it works. Then click **Finish**. We didn't finish here. We should navigate back to blob storage setup: using left pane click **SuperStoreOrders:**



* AzureSqlDWGM
* SuperStoreOrders

Then navigate to **Connection** tab where we will get additional options. We should specify delimiter Tab (\t). We don't have Escape character and Quote character.



Linked service * AzureBlobStorageWorkshop

File path * clouddwinaday / Directory / Sample - Superstore -

Compression type: none

Column delimiter: Tab (\t)
□ Edit

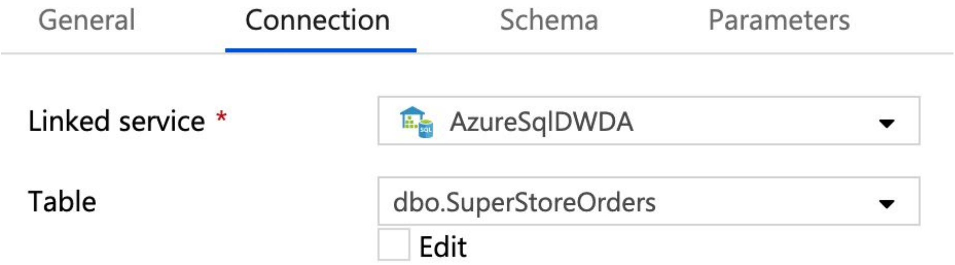
Row delimiter: Auto detect (\r,\n, or \r\n)
□ Edit

Encoding: Default(UTF-8)

Escape character: No escape character
□ Edit

Quote character: No quote character
□ Edit

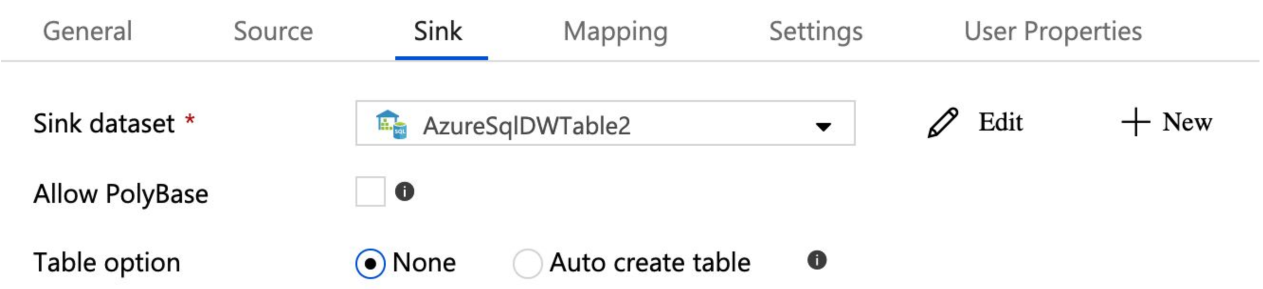
In addition, back at **AsureSqlDW** (use pane on the left to navigate there) we should specify target table that we created previously. Click Edit. Choose the Connection tab and choose the table.



Linked service * AzureSqlDWDA

Table: dbo.SuperStoreOrders
□ Edit

Now navigate to **Pipeline** using tabs at the top**.** By default, ADF is using PolyBase for data loading, make sure to unmark the box:

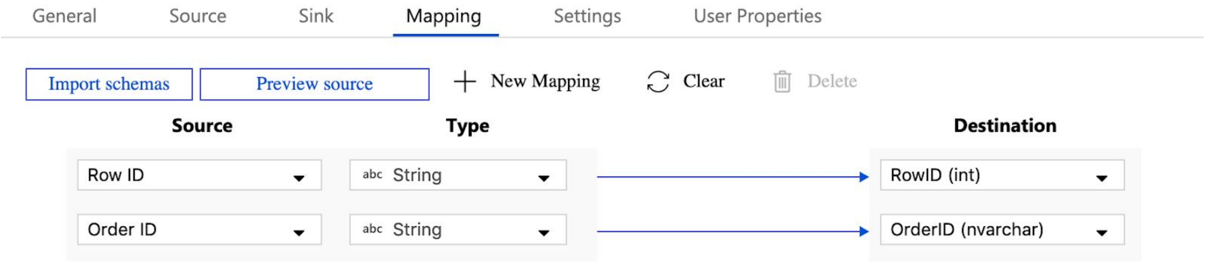


Sink dataset * AzureSqlDWTable2

Allow PolyBase □

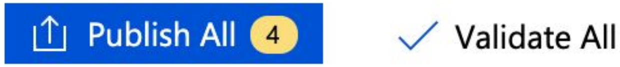
Table option ● None ○ Auto create table

6. Next step is Map source with sink (target). We should click on Import Schemas:

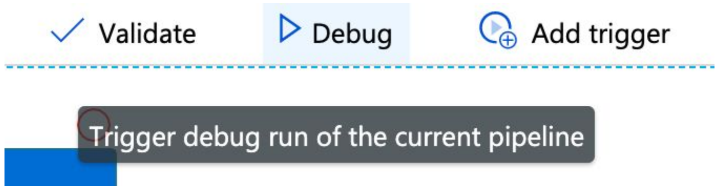


Source	Type	Destination
Row ID	abc String	RowID (int)
Order ID	abc String	OrderID (nvarchar)

7. We have Source, Sink (Target) and Mapping. We are good to go. Let's Validate all and Publish All:



8. We can run our job using Trigger (Add Trigger) or Debug. Let's click Debug. It will launch our job. In case of failure, it will show us failure logs.



Trigger debug run of the current pipeline

It will start the job and load the data into the table.

9. We can see the data by running the following query in Azure Data Studio

   ```
   select City, SUM (Sales) as Sales_Amount
   from dbo.SuperStoreOrders
   group by City
   order by City desc
   ```

   And find out sales for a particular city.

Once again, pause the DW by going to Azure Portal.

Exercise 3: Load data using PolyBase

In this exercise, we will load data into Azure SQL DW using SQL with Polybase. For that **Resume** DW from Azure Portal

1. Let’s start from creating external schema and table that can be used as input for further data transformations. First, we will create a master key if it is not yet created.

   The database master key is a symmetric key used to protect the private keys of certificates and asymmetric keys that are present in the database.

   Try to create master key if it isn't available

   ```
   CREATE MASTER KEY;
   ```

   If it exists, it is ok.

2. Create Credentials for BLOB access using access key (secret). This command already has the key. It is available in Azure Storage menu and we can use it for programmatic access of data.

```
CREATE DATABASE SCOPED CREDENTIAL CLOUDDW WITH IDENTITY = 'user',
SECRET = 'XZC5UyU70XtYz0BsRCpqoMczf3iJMbwBnB4kaQPHFie3oGlBDc63/wLyUEPvciJYnR3w7q5bs+0Twq+4c3uTXg==';
```

3. Create external data source with our credentials:

```
CREATE EXTERNAL DATA SOURCE LogsStorage WITH (
TYPE = HADOOP,
LOCATION =
'wasbs://logs@rydstrorage.blob.core.windows.net',
CREDENTIAL = CLOUDDW
);
```

4. Specify the file format of raw data.

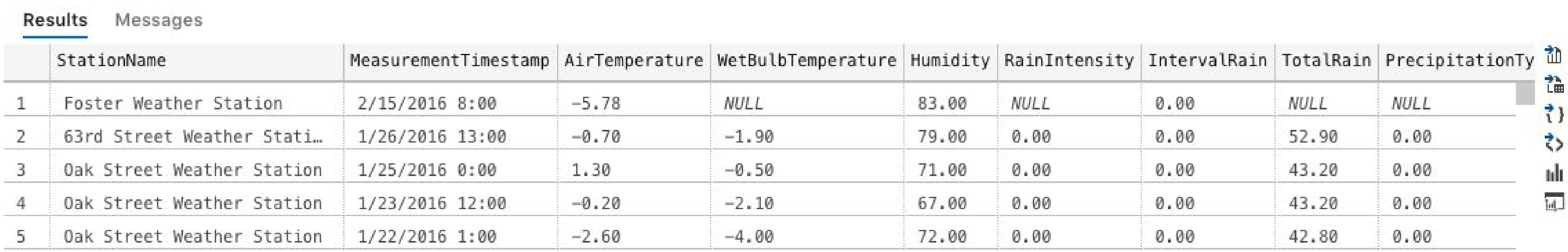
```
CREATE EXTERNAL FILE FORMAT TextFile WITH (
FORMAT_TYPE = DelimitedText, FORMAT_OPTIONS (FIELD_TERMINATOR = ',')
);
```

5. Create external table:

```
CREATE EXTERNAL TABLE dbo.BeachSensorsExternal (
StationName VARCHAR(50) NOT NULL,
MeasurementTimestamp VARCHAR(50) NOT NULL,
AirTemperature DECIMAL(9,2) NULL,
WetBulbTemperature DECIMAL(9,2) NULL,
Humidity DECIMAL(9,2) NULL,
RainIntensity DECIMAL(9,2) NULL,
IntervalRain DECIMAL(9,2) NULL,
TotalRain DECIMAL(9,2) NULL,
PrecipitationType DECIMAL(9,2) NULL,
WindDirection DECIMAL(9,2) NULL,
WindSpeed DECIMAL(9,2) NULL,
MaximumWindSpeed DECIMAL(9,2) NULL,
BarometricPressure DECIMAL(9,2) NULL,
SolarRadiation DECIMAL(9,2) NULL,
Heading DECIMAL(9,2) NULL,
BatteryLife DECIMAL(9,2) NULL,
MeasurementTimestampLabel VARCHAR(50) NOT NULL,
MeasurementID VARCHAR(100) NOT NULL
)
WITH (
LOCATION='/',
DATA_SOURCE=LogsStorage, FILE_FORMAT=TextFile
);
```

6. Now we can query data that lives in BLOB. Let’s test:

```
select * from dbo.BeachSensorsExternal;
```



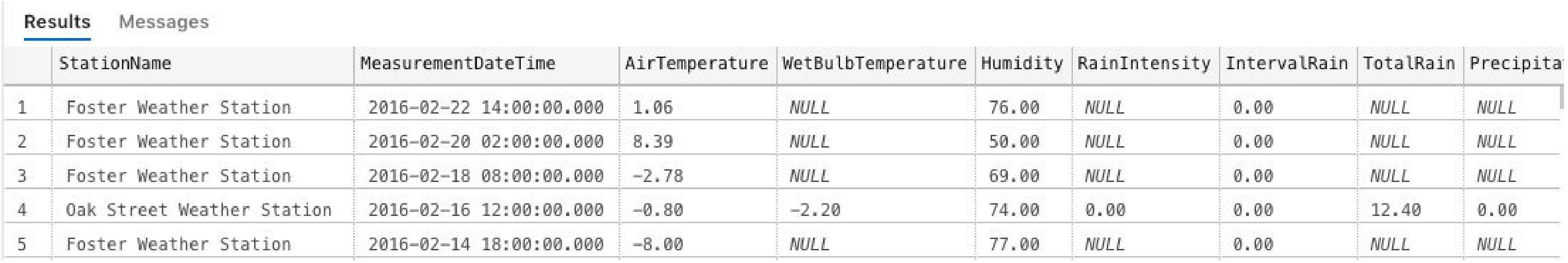
	StationName	MeasurementTimestamp	AirTemperature	WetBulbTemperature	Humidity	RainIntensity	IntervalRain	TotalRain	PrecipitationTy
1	Foster Weather Station	2/15/2016 8:00	-5.78	NULL	83.00	NULL	0.00	NULL	NULL
2	63rd Street Weather Stati…	1/26/2016 13:00	-0.70	-1.90	79.00	0.00	0.00	52.90	0.00
3	Oak Street Weather Station	1/25/2016 0:00	1.30	-0.50	71.00	0.00	0.00	43.20	0.00
4	Oak Street Weather Station	1/23/2016 12:00	-0.20	-2.10	67.00	0.00	0.00	43.20	0.00
5	Oak Street Weather Station	1/22/2016 1:00	-2.60	-4.00	72.00	0.00	0.00	42.80	0.00

7. Now that we have created the external table containing the raw data, you will create a second table in our DW, transform the data, and load it.

```
CREATE TABLE dbo.BeachSensor
WITH (
DISTRIBUTION = ROUND_ROBIN,
CLUSTERED COLUMNSTORE INDEX
) AS
SELECT
StationName,
CAST(MeasurementTimestamp as DATETIME) AS MeasurementDateTime,
AirTemperature,
WetBulbTemperature,
Humidity,
RainIntensity,
IntervalRain,
TotalRain,
PrecipitationType,
WindDirection,
WindSpeed,
MaximumWindSpeed,
BarometricPressure,
SolarRadiation,
Heading,
BatteryLife
FROM dbo.BeachSensorsExternal;
SELECT COUNT(*) FROM dbo.BeachSensor;
```

8. We can query the tables that was just created:

```
SELECT * FROM dbo.BeachSensor;
```



	StationName	MeasurementDateTime	AirTemperature	WetBulbTemperature	Humidity	RainIntensity	IntervalRain	TotalRain	Precipita
1	Foster Weather Station	2016-02-22 14:00:00.000	1.06	NULL	76.00	NULL	0.00	NULL	NULL
2	Foster Weather Station	2016-02-20 02:00:00.000	8.39	NULL	50.00	NULL	0.00	NULL	NULL
3	Foster Weather Station	2016-02-18 08:00:00.000	-2.78	NULL	69.00	NULL	0.00	NULL	NULL
4	Oak Street Weather Station	2016-02-16 12:00:00.000	-0.80	-2.20	74.00	0.00	0.00	12.40	0.00
5	Foster Weather Station	2016-02-14 18:00:00.000	-8.00	NULL	77.00	NULL	0.00	NULL	NULL

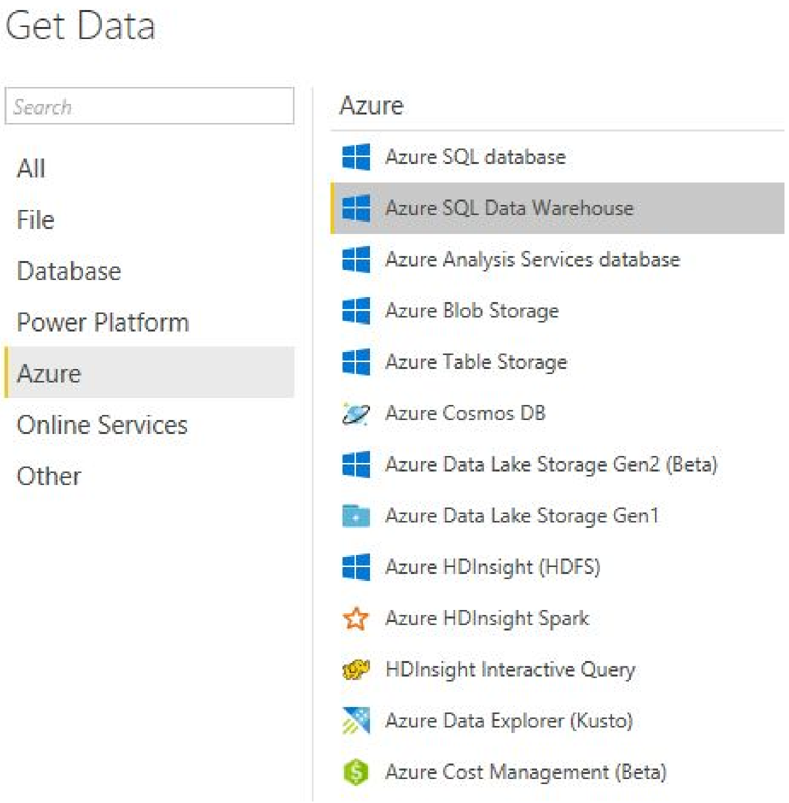
Lab 4: Business Intelligence with Azure SQL DW

In this lab we will learn how to connect Azure SQL DW with Microsoft BI tool - Power BI.

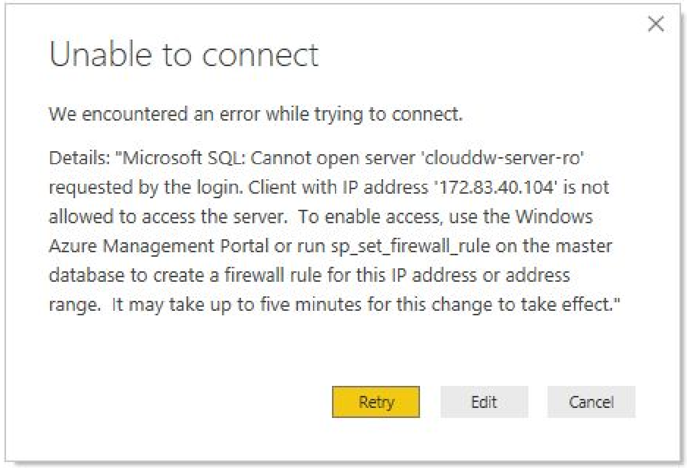
Note: Power BI desktop is available only on Windows. In case if you have Mac OS, you can try to use Azure Data Studio and build somecharts there or use your own BI tool.

Exercise 1: Connecting Azure SQL DW with Power BI - the drag-and-drop way

1. Open Power BI Desktop and click **Get Data.** Power BI Desktop allows direct connection to over 100 sources and list is always growing. Currently we are interested in **Azure SQL Data Warehouse**



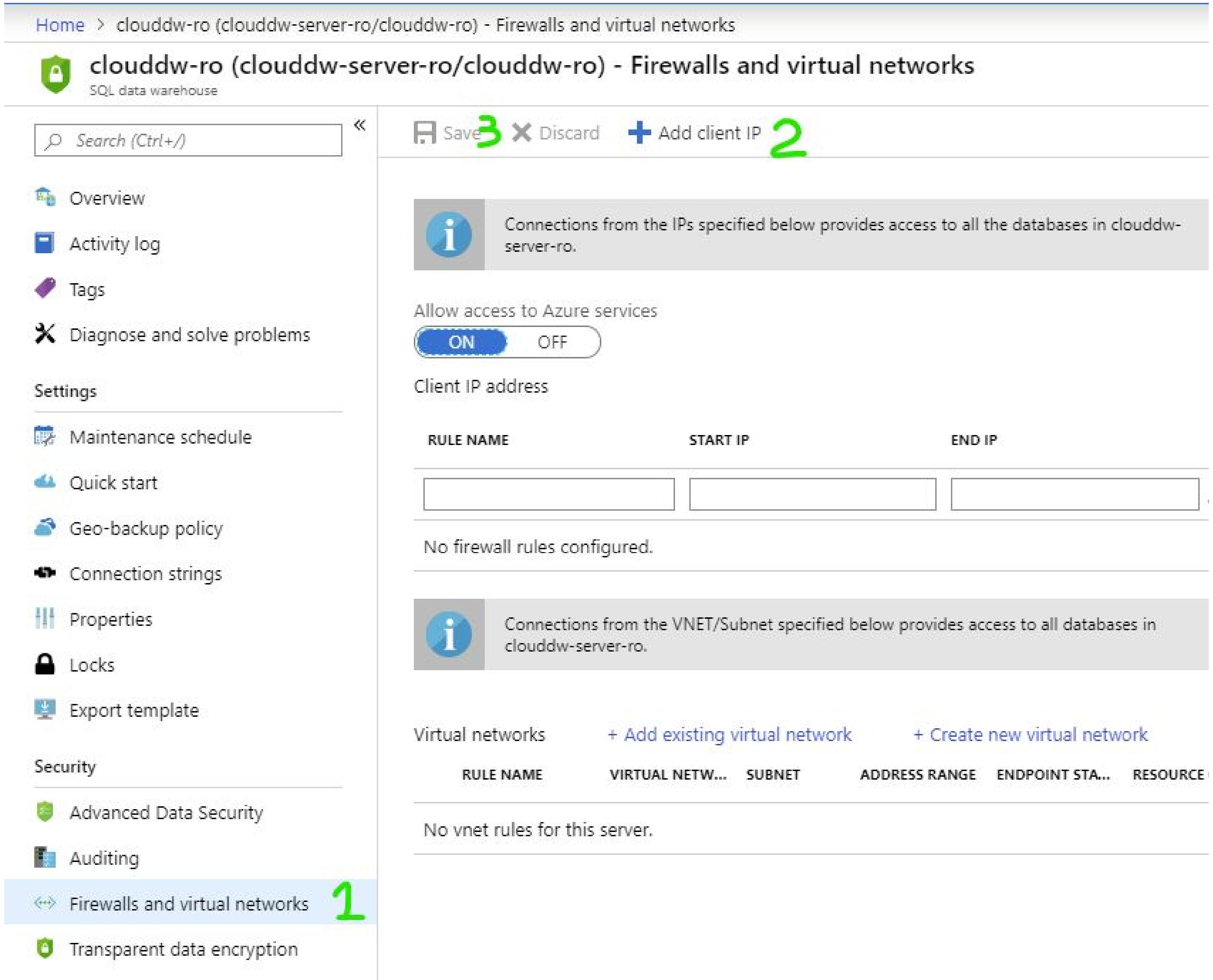
2. If Power BI throws an error message that your IP address is not allowed to access the server, go to https://portal.azure.com,
select your Data warehouse in resources list, open **Firewalls and virtual networks**, click **Add client IP** and **Save.**
This will whitelist your current machine. Alternatively you can specify a range of IP addresses for your organization.



Unable to connect

We encountered an error while trying to connect.

Details: "Microsoft SQL: Cannot open server 'clouddw-server-ro' requested by the login. Client with IP address '172.83.40.104' is not allowed to access the server. To enable access, use the Windows Azure Management Portal or run sp_set_firewall_rule on the master database to create a firewall rule for this IP address or address range. It may take up to five minutes for this change to take effect."



Home > clouddw-ro (clouddw-server-ro/clouddw-ro) - Firewalls and virtual networks

clouddw-ro (clouddw-server-ro/clouddw-ro) - Firewalls and virtual networks
SQL data warehouse

Connections from the IPs specified below provides access to all the databases in clouddw-server-ro.

Allow access to Azure services: ON

Client IP address

RULE NAME	START IP	END IP

No firewall rules configured.

Connections from the VNET/Subnet specified below provides access to all databases in clouddw-server-ro.

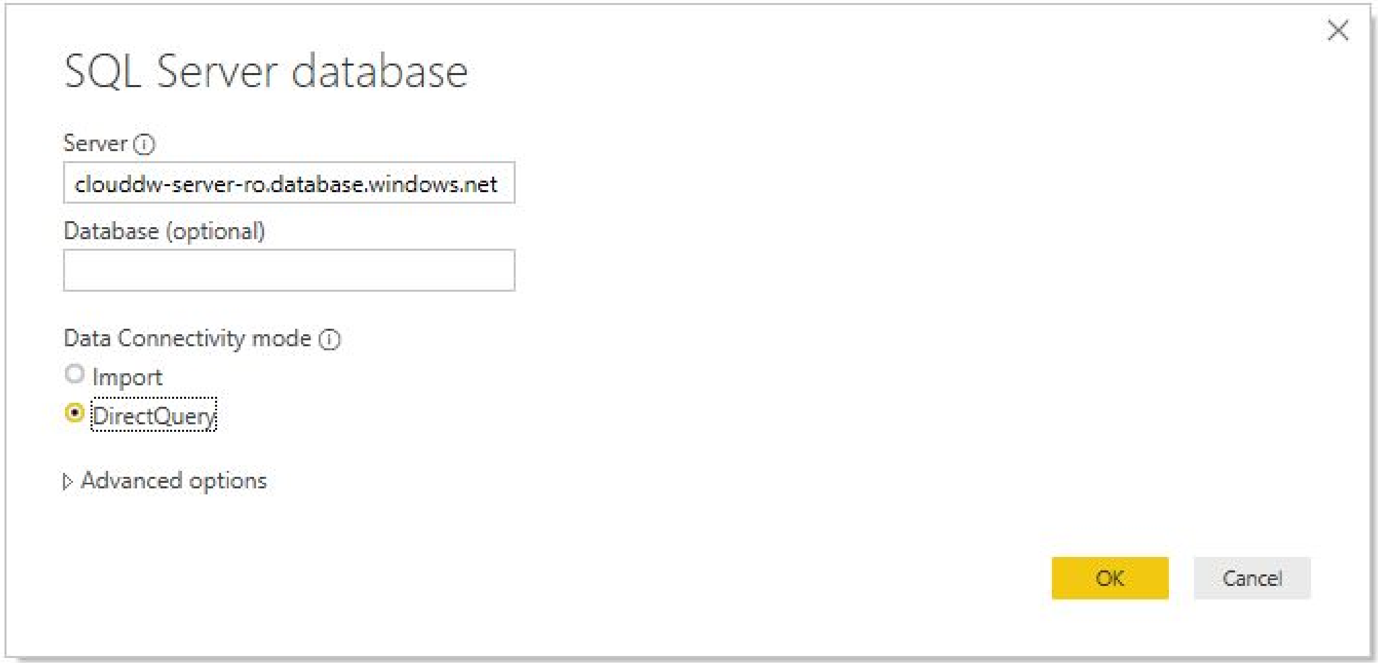
Virtual networks + Add existing virtual network + Create new virtual network

RULE NAME	VIRTUAL NETW...	SUBNET	ADDRESS RANGE	ENDPOINT STA...	RESOURCE

No vnet rules for this server.

.

3. On this stage we are asked to enter server name and choose connectivity mode



SQL Server database

Server ⓘ

clouddw-server-ro.database.windows.net

Database (optional)

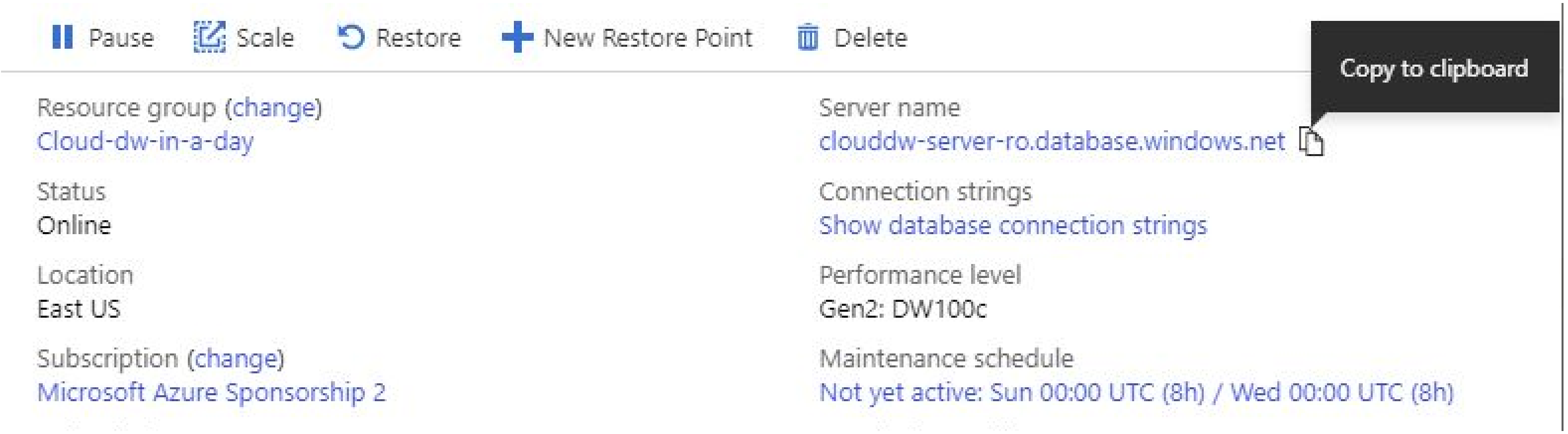
Data Connectivity mode ⓘ

○ Import

● DirectQuery

▷ Advanced options

3a. You can copy server name from Azure Portal.



Resource group (change)
Cloud-dw-in-a-day

Status
Online

Location
East US

Subscription (change)
Microsoft Azure Sponsorship 2

Server name
clouddw-server-ro.database.windows.net

Connection strings
Show database connection strings

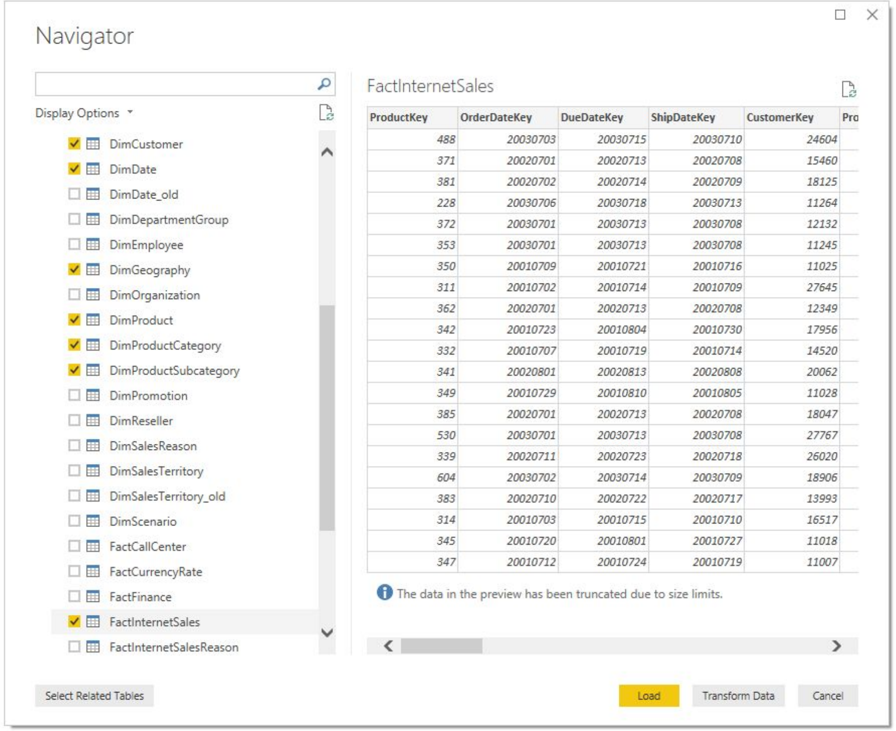
Performance level
Gen2: DW100c

Maintenance schedule
Not yet active: Sun 00:00 UTC (8h) / Wed 00:00 UTC (8h)

3b. Import vs DirectQuery.

Basically import mode loads all dataset in memory, where it is then sliced and diced to create visuals.
DirectQuery, as the name suggests, sends real time queries to the server each time a piece of data is requested for a visual.

Both solutions have pros and cons. Which to choose in each case depends on each case. For small dataset that does not need to be updated frequently like our sample dataset, Import mode will provide the best performance, direct access to tables, ability to use natural language queries and Quick Insights.
For large datasets that are normally used when people need a data warehouse, DirectQuery is the only way to go. It has no limit on the dataset size
Another option would be creating Azure Analysis Service tabular model and connecting it to Power BI. That would be a good choice for multinational organization that has a global data model that is accessed by analysts in different countries to build their BI reports using unified set of fields, measures and relationships.For our workshop let's focus on import mode as the most simple one, then you can try other methods and compare performance and behaviour.



- ☑ DimCustomer
- ☑ DimDate
- □ DimDate_old
- □ DimDepartmentGroup
- □ DimEmployee
- ☑ DimGeography
- □ DimOrganization
- ☑ DimProduct
- ☑ DimProductCategory
- ☑ DimProductSubcategory
- □ DimPromotion
- □ DimReseller
- □ DimSalesReason
- □ DimSalesTerritory
- □ DimSalesTerritory_old
- □ DimScenario
- □ FactCallCenter
- □ FactCurrencyRate
- □ FactFinance
- ☑ FactInternetSales
- □ FactInternetSalesReason

FactInternetSales

ProductKey	OrderDateKey	DueDateKey	ShipDateKey	CustomerKey	Pro
488	20030703	20030715	20030710	24604	
371	20020701	20020713	20020708	15460	
381	20020702	20020714	20020709	18125	
228	20030706	20030718	20030713	11264	
372	20030701	20030713	20030708	12132	
353	20030701	20030713	20030708	11245	
350	20010709	20010721	20010716	11025	
311	20010702	20010714	20010709	27645	
362	20020701	20020713	20020708	12349	
342	20010723	20010804	20010730	17956	
332	20010707	20010719	20010714	14520	
341	20020801	20020813	20020808	20062	
349	20010729	20010810	20010805	11028	
385	20020701	20020713	20020708	18047	
530	20030701	20030713	20030708	27767	
339	20020711	20020723	20020718	26020	
604	20030702	20030714	20030709	18906	
383	20020710	20020722	20020717	13993	
314	20010703	20010715	20010710	16517	
345	20010720	20010801	20010727	11018	
347	20010712	20010724	20010719	11007	

The data in the preview has been truncated due to size limits.

4. Select Tables

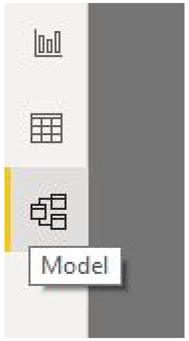
For our purpose let's choose 7 tables:

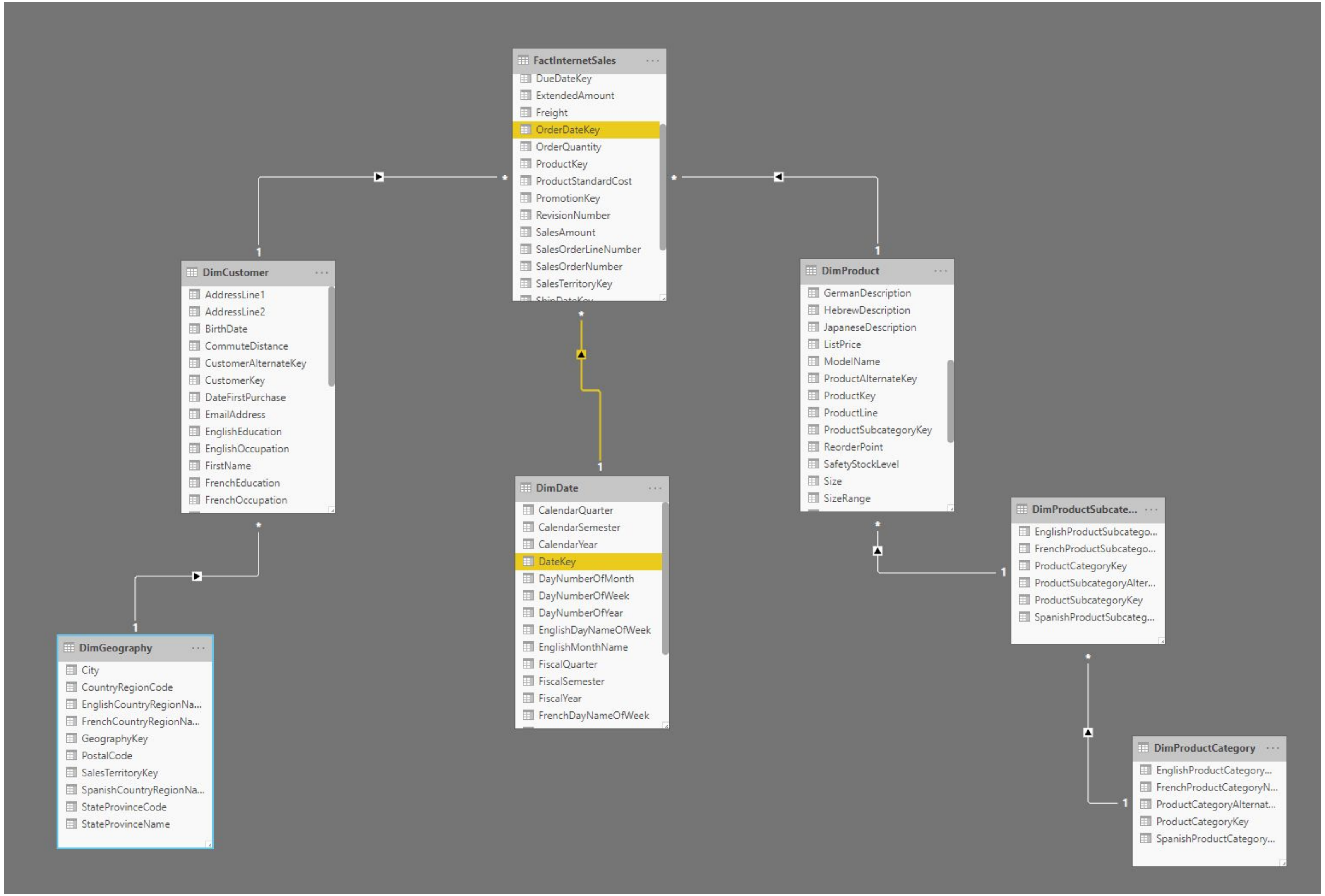
1) DimDate
2) DimCustomer
3) DimGeography
4) DimProduct
5) DimProductCategory
6) DimProductSubcategory
7) FactInternetSales

[Optional] Clicking **Transform Data** opens Power Query editor - integrated data preparation tool that allows you to clean your data, merge and append queries, perform calculations, aggregate values, pivot/unpivot tables and many more. Most operations in Power Query can be performed without any coding! Note that DirectQuery mode doesn't support most transformations in Power Query.

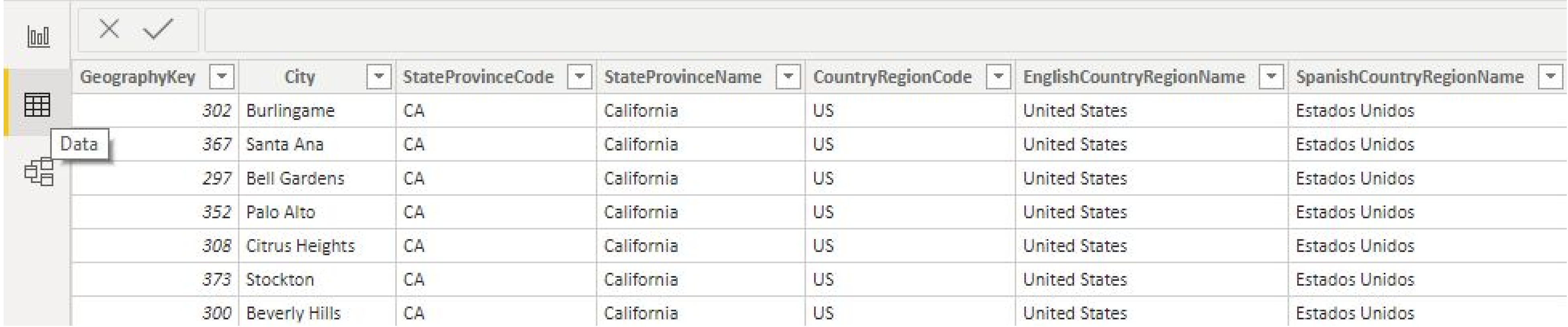
5. Create Relationships.

Power BI automatically detects relationships on load. However. sometimes there are multiple ways to connect tables. Click **Model** view on the left panel. To join 2 tables just drag field from one table to corresponding field in another table. Then select the type of relation and filter direction.





6. Check tables in the **Data** view. They are very helpful to build reports and measures. Notice that you don't have this view if connected Live (DirectQuery or Tabular)

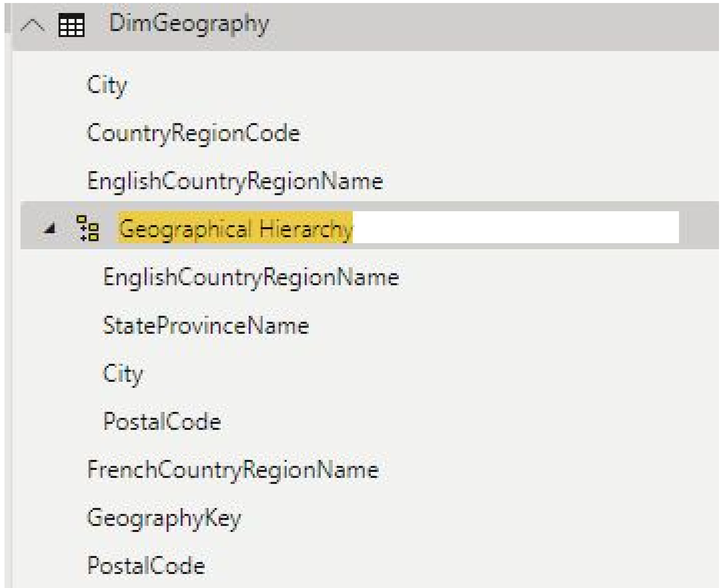


GeographyKey	City	StateProvinceCode	StateProvinceName	CountryRegionCode	EnglishCountryRegionName	SpanishCountryRegionName
302	Burlingame	CA	California	US	United States	Estados Unidos
367	Santa Ana	CA	California	US	United States	Estados Unidos
297	Bell Gardens	CA	California	US	United States	Estados Unidos
352	Palo Alto	CA	California	US	United States	Estados Unidos
308	Citrus Heights	CA	California	US	United States	Estados Unidos
373	Stockton	CA	California	US	United States	Estados Unidos
300	Beverly Hills	CA	California	US	United States	Estados Unidos

7. Select DimGeography table. Notice that fields are hierarchical here
CountryRegionName > StateProvinceName > City > PostalCode
This can be used for powerful drill downs. In this case - by location.

In **Fields** pane Drag **StateProvinceName** over **EnglishCountryRegionName**.
Then drag **City** and **PostalCode** over **EnglishCountryRegionName** Hierarchy.

You can optionally doubleclick hierarchy and give it a better name.

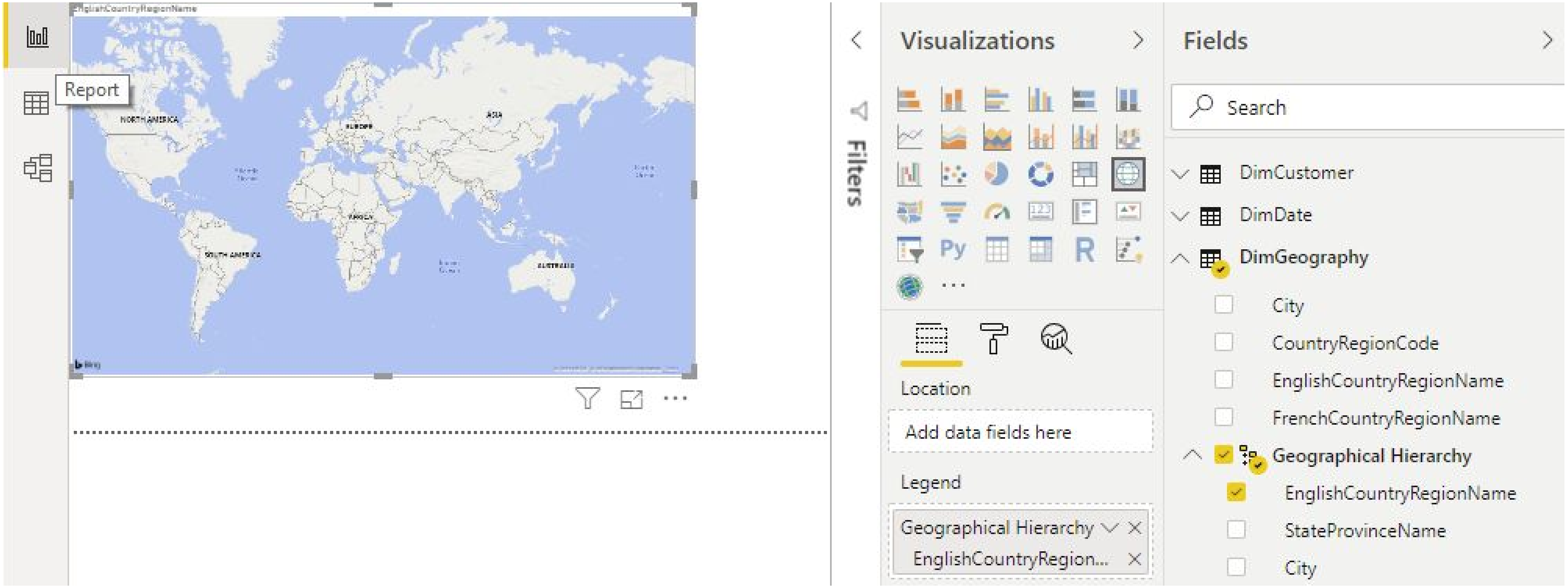


DimGeography
- City
- CountryRegionCode
- EnglishCountryRegionName
- Geographical Hierarchy
  - EnglishCountryRegionName
  - StateProvinceName
  - City
  - PostalCode
- FrenchCountryRegionName
- GeographyKey
- PostalCode

Exercise 2: Building the Dashboard and publishing it in Power BI service

1. Build Visuals.

Go to Reports View and drag our hierarchy to the dashboard. Notice that Power BI recognizes that data is geographical and creates map visual putting Country in the Legend of this visual. Let's add **StateProvinceName**, **City** and **PostalCode** either by dragging them on the map or just by clicking checkboxes, so Power BI will understand that we want to be able to drill down on this hierarchy.



Visualizations

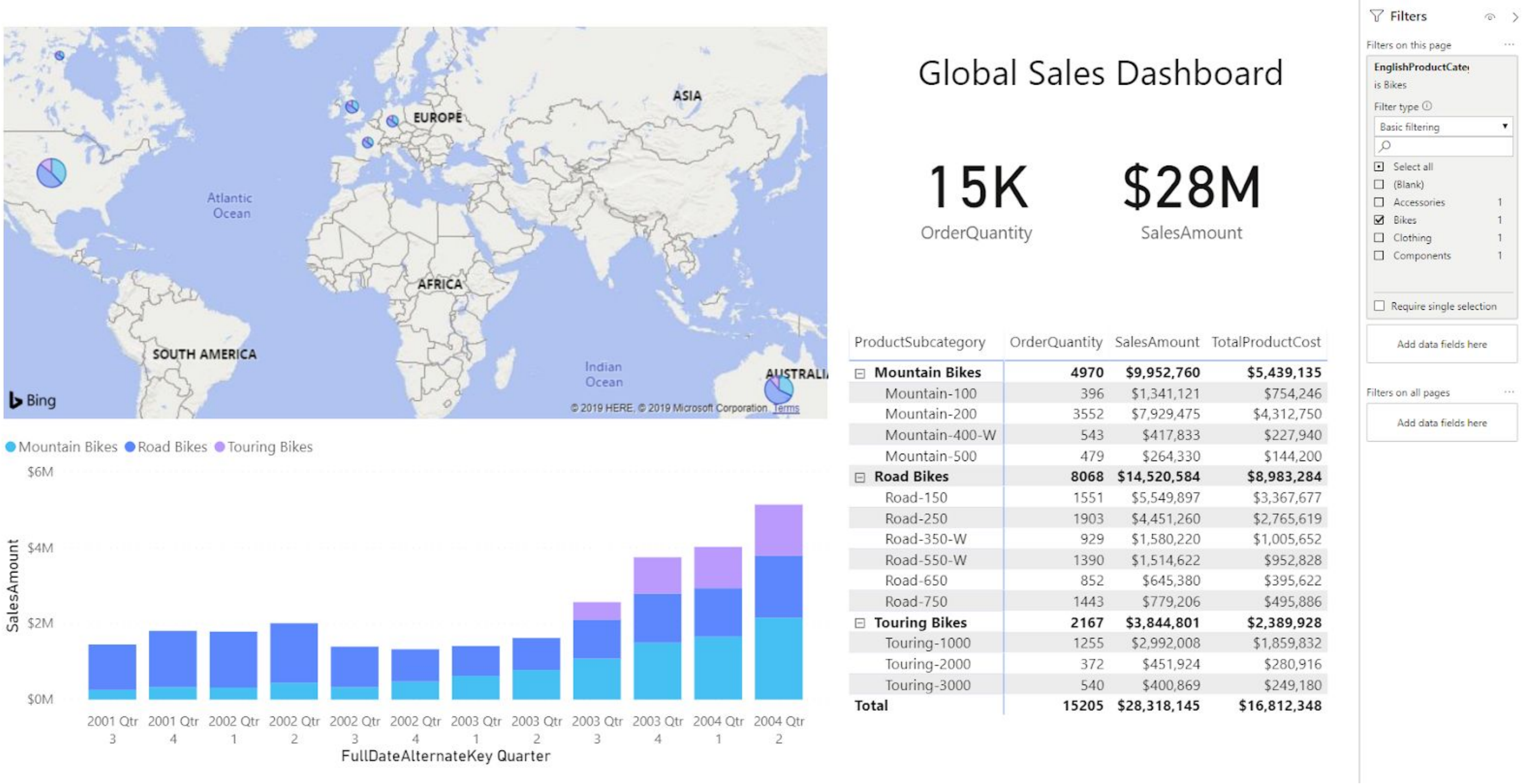
Location: Add data fields here

Legend: Geographical Hierarchy, EnglishCountryRegion...

Fields

- DimCustomer
- DimDate
- DimGeography
  - □ City
  - □ CountryRegionCode
  - □ EnglishCountryRegionName
  - □ FrenchCountryRegionName
  - ☑ Geographical Hierarchy
    - ☑ EnglishCountryRegionName
    - □ StateProvinceName
    - □ City

2. Using drag-and-drop Try to build a dashboard like this or do better. Play with it, try drill down on hierarchies. On the next page you can find fields that was used in each visual,



Global Sales Dashboard

15K
OrderQuantity

$28M
SalesAmount

ProductSubcategory	OrderQuantity	SalesAmount	TotalProductCost
⊟ **Mountain Bikes**	**4970**	**$9,952,760**	**$5,439,135**
Mountain-100	396	$1,341,121	$754,246
Mountain-200	3552	$7,929,475	$4,312,750
Mountain-400-W	543	$417,833	$227,940
Mountain-500	479	$264,330	$144,200
⊟ **Road Bikes**	**8068**	**$14,520,584**	**$8,983,284**
Road-150	1551	$5,549,897	$3,367,677
Road-250	1903	$4,451,260	$2,765,619
Road-350-W	929	$1,580,220	$1,005,652
Road-550-W	1390	$1,514,622	$952,828
Road-650	852	$645,380	$395,622
Road-750	1443	$779,206	$495,886
⊟ **Touring Bikes**	**2167**	**$3,844,801**	**$2,389,928**
Touring-1000	1255	$2,992,008	$1,859,832
Touring-2000	372	$451,924	$280,916
Touring-3000	540	$400,869	$249,180
Total	**15205**	**$28,318,145**	**$16,812,348**

Filters

Filters on this page

EnglishProductCate
is Bikes

Filter type ⓘ
Basic filtering ▾

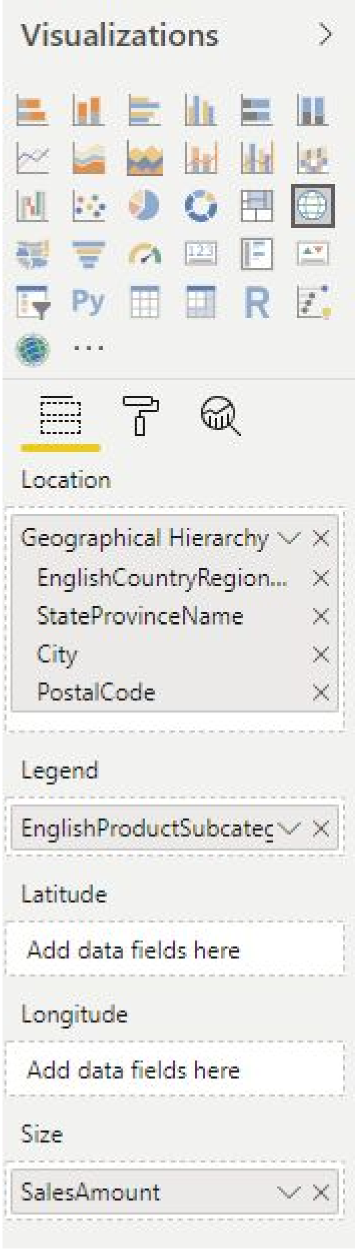
☒ Select all
☐ (Blank)
☐ Accessories 1
☑ Bikes 1
☐ Clothing 1
☐ Components 1

☐ Require single selection

Add data fields here

Filters on all pages

Add data fields here



Visualizations

Location

Geographical Hierarchy

EnglishCountryRegion...

StateProvinceName

City

PostalCode

Legend

EnglishProductSubcateg

Latitude

Add data fields here

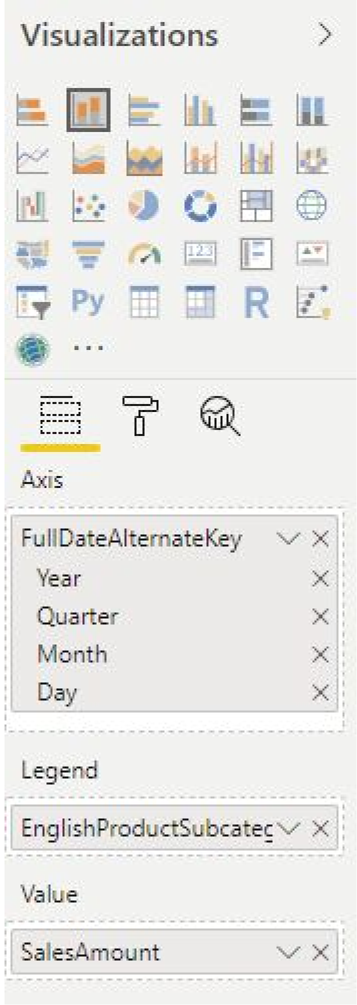
Longitude

Add data fields here

Size

SalesAmount

Map



Visualizations

Axis

FullDateAlternateKey

Year

Quarter

Month

Day

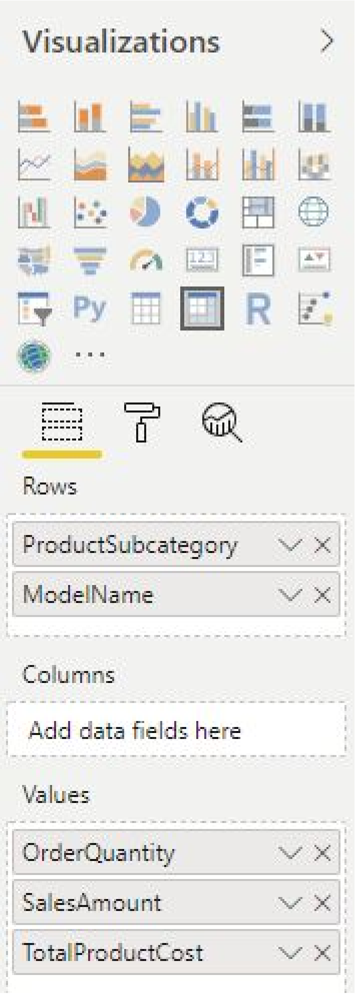
Legend

EnglishProductSubcateg

Value

SalesAmount

Column Chart



Visualizations

Rows

ProductSubcategory

ModelName

Columns

Add data fields here

Values

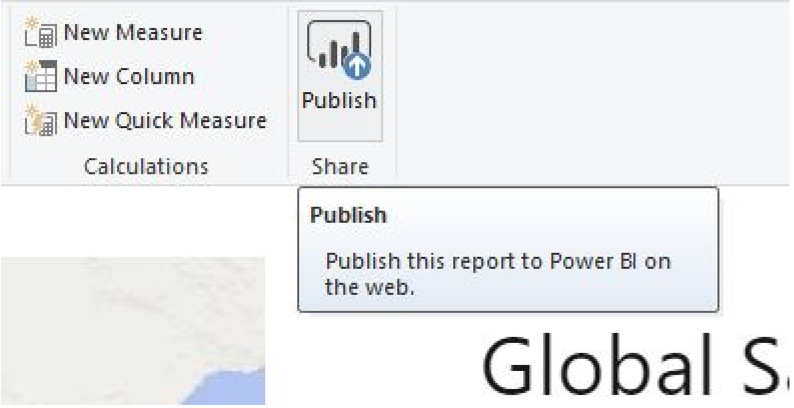
OrderQuantity

SalesAmount

TotalProductCost

Table

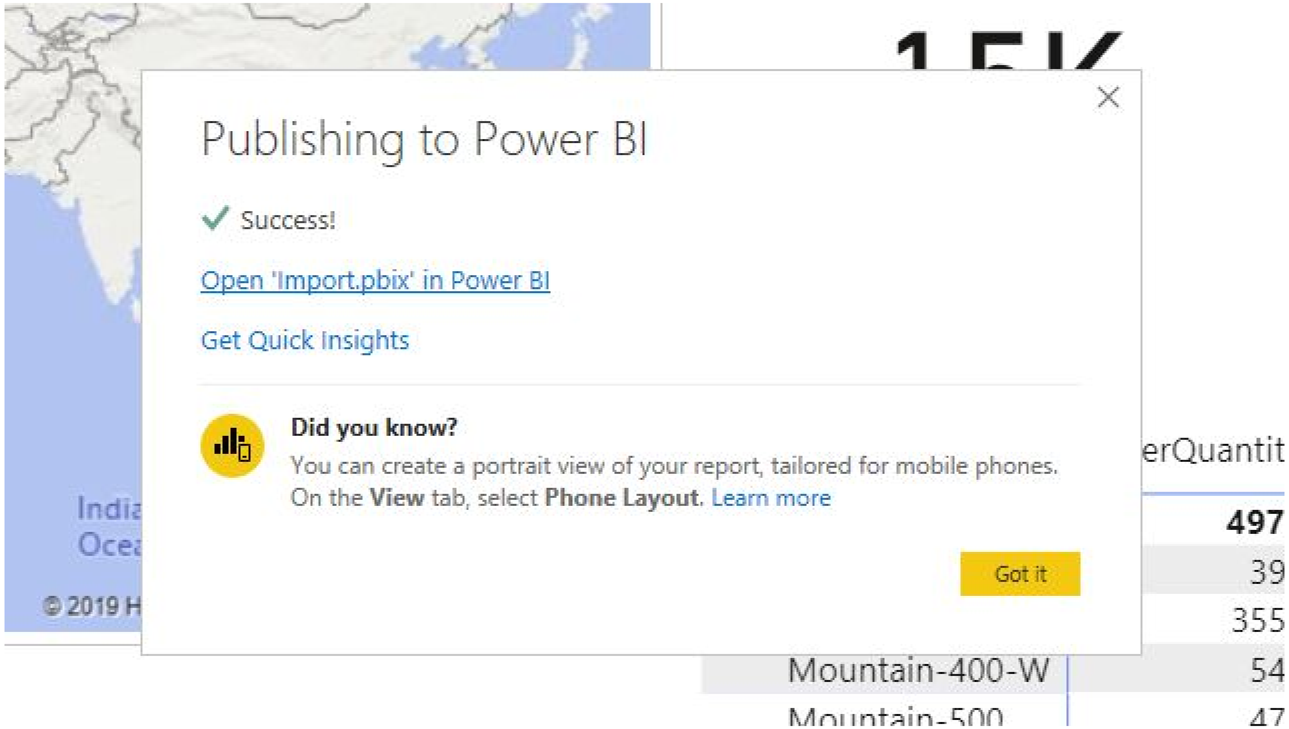
3. Save your report and publish it on Power BI Service



Publish

Publish this report to Power BI on the web.

4. Select **My Workspace** and wait for it to finish uploading.



Publishing to Power BI

✓ Success!

Open 'Import.pbix' in Power BI

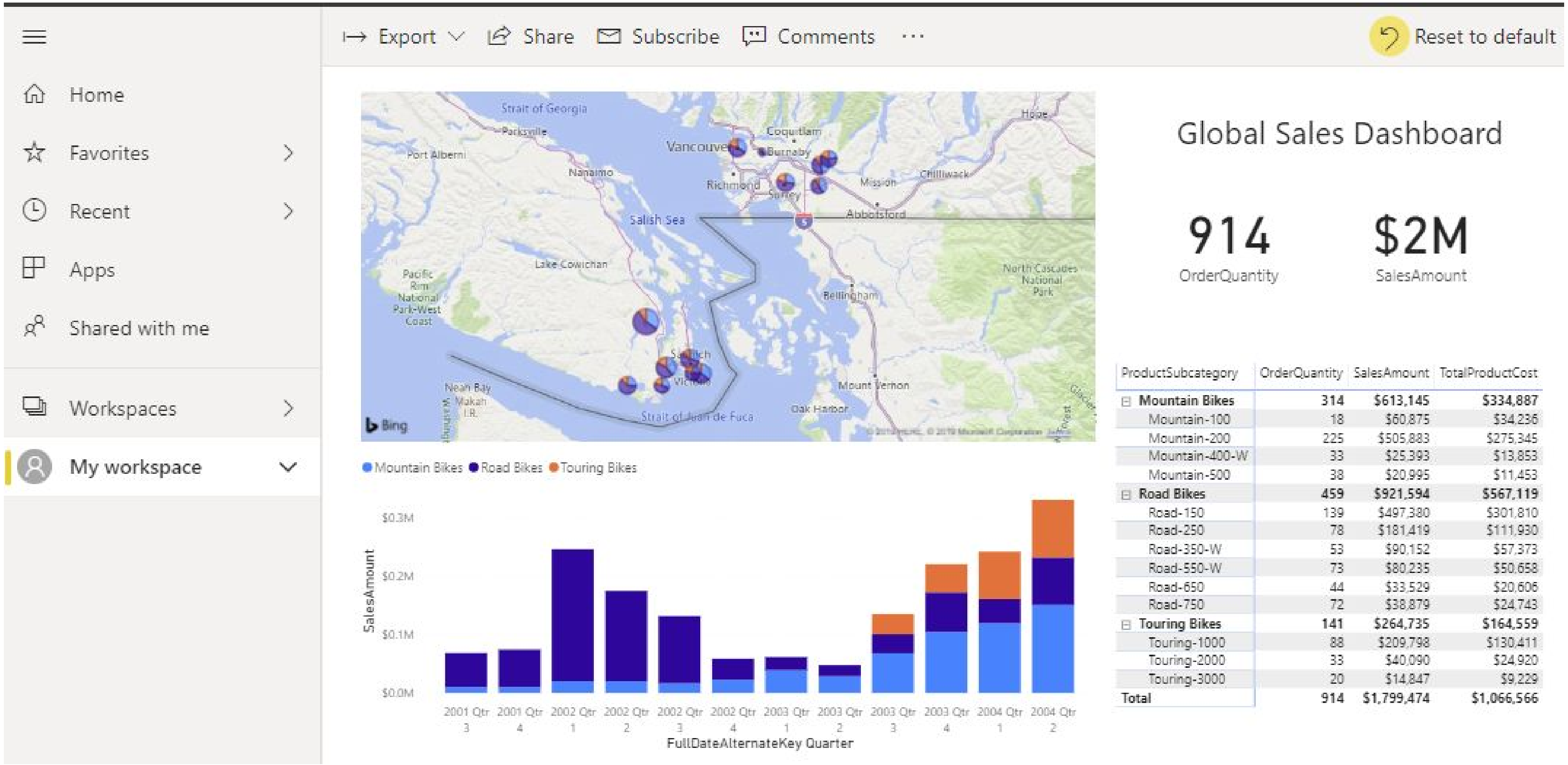
Get Quick Insights

Did you know?

You can create a portrait view of your report, tailored for mobile phones. On the **View** tab, select **Phone Layout**. Learn more

Got it

5. Open report in Power BI Service (Sign in to your Microsoft account if asked by the system)



Global Sales Dashboard

914
OrderQuantity

$2M
SalesAmount

ProductSubcategory	OrderQuantity	SalesAmount	TotalProductCost
⊟ **Mountain Bikes**	**314**	**$613,145**	**$334,887**
Mountain-100	18	$60,875	$34,236
Mountain-200	225	$505,883	$275,345
Mountain-400-W	33	$25,393	$13,853
Mountain-500	38	$20,995	$11,453
⊟ **Road Bikes**	**459**	**$921,594**	**$567,119**
Road-150	139	$497,380	$301,810
Road-250	78	$181,419	$111,930
Road-350-W	53	$90,152	$57,373
Road-550-W	73	$80,235	$50,658
Road-650	44	$33,529	$20,606
Road-750	72	$38,879	$24,743
⊟ **Touring Bikes**	**141**	**$264,735**	**$164,559**
Touring-1000	88	$209,798	$130,411
Touring-2000	33	$40,090	$24,920
Touring-3000	20	$14,847	$9,229
Total	**914**	**$1,799,474**	**$1,066,566**

From Power BI Service you can distribute your report in various ways (requires pro license), export to various formats, download data, print out, etc.

Enjoy your new skills!