

Data Warehouse

A traditional sql database would have a bunch of tables usually with primary keys and foreign keys defining the relationships between them. The database is also normalized so that there isn't duplicate data in the table. This would usually be the structure for transactional purposes where there needs to be high write capacity in the database.

For Analytical purposes we need a Data Warehouse. The tables would be in the form of fact tables and dimension tables.

Fact Tables

- These are measurements or metrics that correspond to facts
- For example, a sales table would have records of all the sales that have been made
- The sales data are facts that sales have actually been made

Dimension Tables

- These help provide some sort of context to the facts that are being presented
- For example, what are the products that were sold, who were the customers who bought the product, etc. is context about facts presented in the fact table

For the dimension tables, it is not necessary that the data would only come from the sql database. It could come from various sources.

So, let's say if there are two or more sources for a particular dimension table, then we would make use of something called a surrogate key. It helps uniquely identify each row in the table. So even if there are matching ids from two different sources, the surrogate key would never be the same for the two and hence every row would get its own unique identifier that way.

We can make use of the identity column feature in Azure Synapse for tables to generate the unique id.

Also, its best practice to not have null values in your dimension tables.

Creating a new Data Warehouse

Download [this](#) bacpac file so that we can use the data to create a warehouse. Upload this file to the same data lake where we have our other files that we have been using (Log.csv , parquet files). Create a subdirectory in the raw folder by the name of sql and upload the bacpac file there.

Begin by heading over to your SQL database server. In my case it is named azure203lab. Click on the import database button. Click on select backup. It would load up all the places from where it can load a backup, select your bacpac file from the datalake. Configure the database according to your needs, give a database name finally enter your password for the server admin log in.

Import database ...
azure203lab

Subscription *
Azure subscription 1

Storage (Premium not supported) *
data.bacpac
datalake203prep/sql
[Select backup](#)

Pricing tier * ⓘ
Standard S0
10 DTUs, 1 GB storage
[Configure database](#)

Database name
adventureworks ✓

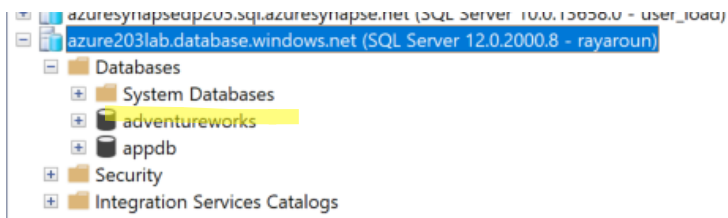
Collation * ⓘ
SQL_Latin1_General_CP1_CI_AS

Authentication type
SQL Server

Server admin login *
rayaroun

*Password
..... ✓

Now we can go to the SSMS check out our database in the sql server.



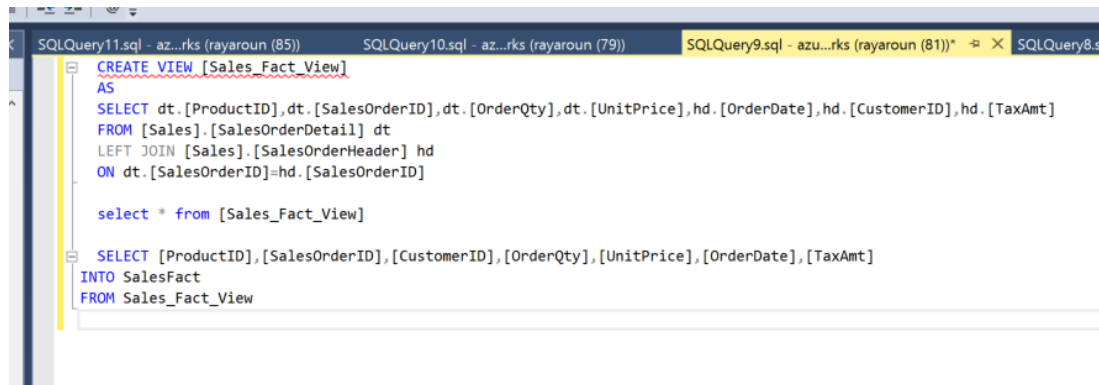
Building a fact table

These are usually large in size and contain measurable facts. These would contain the primary keys used in the dimension tables.

We would be using [this](#) scrip to create our fact table. Right click on adventureworks and query it. We are creating a fact table based on the SaledOrderDetail table and the SaledOrderHeader table.

We are creating the Fact table in the SQL database itself but the right approach is to create the table in Azure Synapse Directly.

We are first creating a view and then creating a table based on that view.



This is how it looks finally –

	ProductID	SalesOrderID	CustomerID	OrderQty	UnitPrice	OrderDate	TaxAmt
1	776	43659	29825	1	2024.994	2011-05-31 00:00:00.000	1971.5149
2	777	43659	29825	3	2024.994	2011-05-31 00:00:00.000	1971.5149
3	778	43659	29825	1	2024.994	2011-05-31 00:00:00.000	1971.5149
4	771	43659	29825	1	2039.994	2011-05-31 00:00:00.000	1971.5149
5	772	43659	29825	1	2039.994	2011-05-31 00:00:00.000	1971.5149

So it has facts as to what product was ordered, what was the sales id, how many products were ordered etc. Now we can have corresponding dimension tables that would have information about products based on product ids, costumers based on customer id, etc.

Building a Dimension table

For this part we would be using [this](#) script. Like previously mentioned, the dimension tables would play the role of supporting the information in the fact tables so like there should be a dimension table that would have customer information based on customer id, we create a customer dimension table.

We create this customer table from the Customer table joined with the Store table.

```
SQLQuery11.sql - az...rks (rayaroun (85))  SQLQuery10.sql - az...rks (rayaroun (79))  SQLQuery9.sql - azu...rks (rayaroun (81))
CREATE VIEW Customer_view
AS
SELECT ct.[CustomerID],ct.[StoreID],st.[BusinessEntityID],st.[Name] as StoreName
FROM [Sales].[Customer] as ct
LEFT JOIN [Sales].[Store] as st
ON ct.[StoreID]=st.[BusinessEntityID]
WHERE st.[BusinessEntityID] IS NOT NULL

SELECT [CustomerID],[StoreID],[BusinessEntityID],StoreName
INTO DimCustomer
FROM Customer_view
```

Looks like –

	CustomerID	StoreID	BusinessEntityID	StoreName
1	1	934	934	A Bike Store
2	2	1028	1028	Progressive Sports
3	3	642	642	Advanced Bike Components
4	4	932	932	Modular Cycle Systems
5	5	1026	1026	Metropolitan Sports Supply
6	6	644	644	Aerobic Exercise Companv

Next we create a product dimension table.

```
SQLQuery11.sql - az...rks (rayaroun (85))  SQLQuery10.sql - az...rks (rayaroun (79))  SQLQuery9.sql - azu...rks (rayaroun (81))*  SQLQuery8.sql - azu...rks (rayaroun (67))*
CREATE VIEW Product_view
AS
SELECT prod.[ProductID],prod.[Name] as ProductName,prod.[SafetyStockLevel],model.[ProductModelID],model.[Name] as ProductModelName,category.[ProductSubcategoryID],category.[Name] as ProductSubCategoryName
FROM [Production].[Product] prod
LEFT JOIN [Production].[ProductModel] model ON prod.[ProductModelID] = model.[ProductModelID]
LEFT JOIN [Production].[ProductSubcategory] category ON prod.[ProductSubcategoryID]=category.[ProductSubcategoryID]
WHERE prod.[ProductModelID] IS NOT NULL

SELECT [ProductID],[ProductModelID],[ProductSubcategoryID],ProductName,[SafetyStockLevel],ProductModelName,ProductSubCategoryName
INTO DimProduct
FROM Product_view
```

Which looks like –

	ProductID	ProductModelID	ProductSubcategoryID	ProductName	SafetyStockLevel	ProductModelName	ProductSubCategoryName
1	680	6	14	HL Road Frame - Black, 58	500	HL Road Frame	Road Frames
2	706	6	14	HL Road Frame - Red, 58	500	HL Road Frame	Road Frames
3	707	33	31	Sport-100 Helmet, Red	4	Sport-100	Helmets
4	708	33	31	Sport-100 Helmet, Black	4	Sport-100	Helmets
5	709	18	23	Mountain Bike Socks, M	4	Mountain Bike Socks	Socks

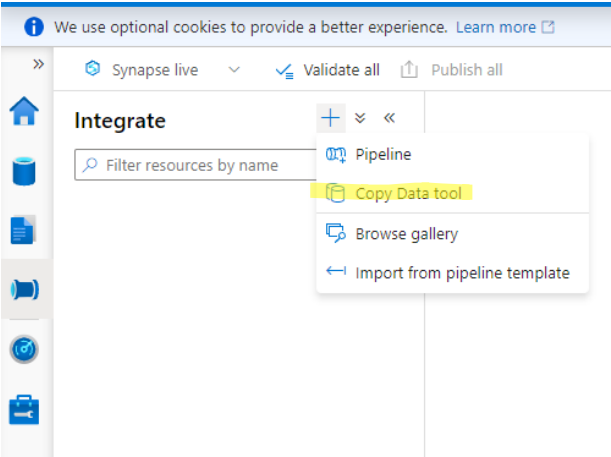
Transferring these table to the Dedicated SQL Pool

We can perform the load operation using the integrate feature that is present in the Synapse Studio. This is done using creating pipelines which are based on Azure Data Factory. For now we would be creating a pipeline that would copy our data from the sql tables as it is onto our dedicated sql pool.

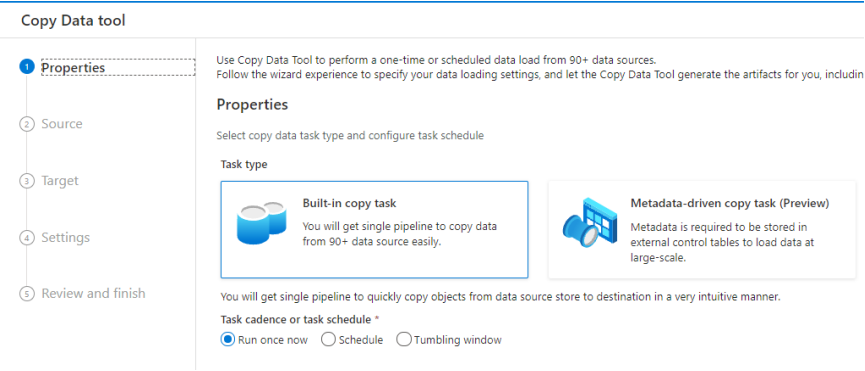
For this part we would be using [this](#) script.

Connect to your dedicated sql pool on your SSMS and Create the SalesFact table followed by DimCustomer followed by DimProduct.

Head over to the Synapse Studio, go to the integrate section from the left-hand menu. Click on the plus sign and select Copy Data tool.



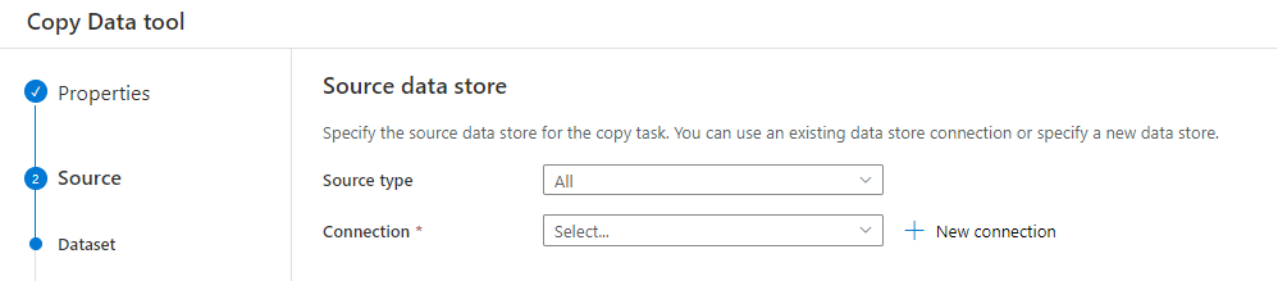
We see the following screen – we select Run once now



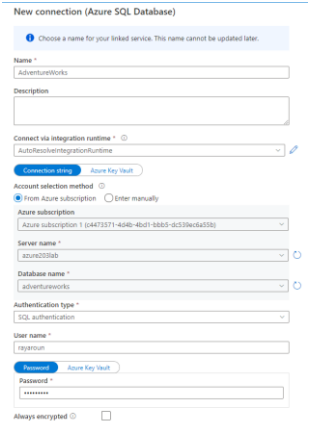
Note - Whenever we are building a pipeline where the target is on Azure Synapse, then there needs to be a storage account set up to be used as a staging area. This storage account would take data from the source and perform any sort of work required on the data and then send the data to Synapse.

For this purpose, we create a new container in our Gen2 storage account and name it synapse.

In Synapse studio after click next we get – we select new connection here.



In the new connection screen we go to the Azure tab, select Azure SQL Database. Then we name the connection, select our subscription, server name, database name, authentication as sql authentication and finally enter our credential. We test the connection and then hit create.



On the next screen after creating the connection, you would be asked the tables that you want to use. We select our fact table and the two dimension tables.

Source data store

Specify the source data store for the copy task. You can use an existing data store connection or specify a new data store.

Source type: All

Connection *: AdventureWorks

Integration runtime *: AutoResolveIntegrationRuntime

Source tables: Existing tables

Filter by name... Show views Refresh Showing 74 out of 74 tables, 0 out of 23 views (3 selected)

Select all

dbo.AWBuildVersion

dbo.DatabaseLog

dbo.DimCustomer

dbo.DimProduct

dbo.ErrorLog

dbo.SalesFact

For the next screen we have the again a new connection for our target which is Azure Synapse. We click new connection and select Azure Synapse Analytics from the Azure tab. Enter a name for the connection and select other details just like before.

New connection (Azure Synapse Analytics)

Choose a name for your linked service. This name cannot be updated later.

Name *: AzureSynapseAnalytics

Description

Connect via integration runtime *: AutoResolveIntegrationRuntime

Connection string: Azure Key Vault

Account selection method: From Azure subscription

Azure subscription: Azure subscription 1 (d4473571-4d4b-4bd1-bbb5-dc539eca55b)

Server name *: azuresynapsedp203 (Synapse workspace)

Database name *: dp203

SQL pool *: Select SQL pool

Authentication type *: SQL authentication

User name *: sqladminuser

Password *:

On the next screen you can see it has automatically mapped the tables in the source to the tables in the target because the table names are the same.

Microsoft Azure | Synapse Analytics | azuresynapsedp203

Copy Data tool

Destination data store

Specify the destination data store for the copy task. You can use an existing data store connection or specify a new data store.

Target type: All

Connection *: AzureSynapseAnalytics

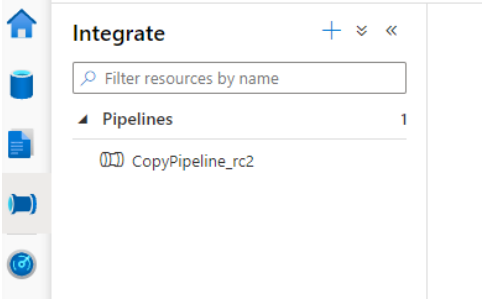
Integration runtime *: AutoResolveIntegrationRuntime

Source	Target
dbo.DimCustomer	dbo.DimCustomer
dbo.DimProduct	dbo.DimProduct
dbo.SalesFact	dbo.SalesFact

Click next. On the next screen we can even do a column based mapping. We again click next.

On the next screen under Staging setting, we would have to like a staging area like previously mentioned. Click on New.

Enter a name for the service, select Azure Data Lake Storage Gen2 in the type, select the Data lake in storage name and click create. On the next screen in storage path browse to the synapse container and select it. In advanced we can see it is using the Polybase technique to copy it. We can select bulk insert as we are copying everything. Click next and finish. It would show up as –



If you go to the monitor tab, you can also see that the pipeline has also succeeded in running.

Pipeline name	Run start	Run end	Duration	Triggered by	Status	Error	Run	Parameters	Annotations	Run ID
CopyPipeline_rc2	10/20/21, 2:17:31 PM	10/20/21, 2:18:19 PM	00:00:48	Manual trigger	Succeeded		Original			9b96e1e1-190a-47c

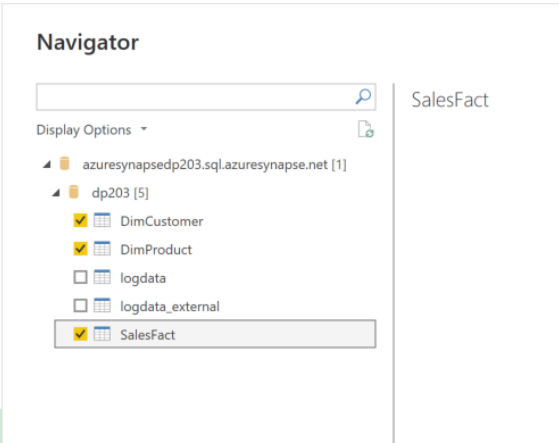
Meaning the data has been loaded. If we go to the SSMS and select * on the fact or dimension tables, the data would show up.

The fact table shows up as –

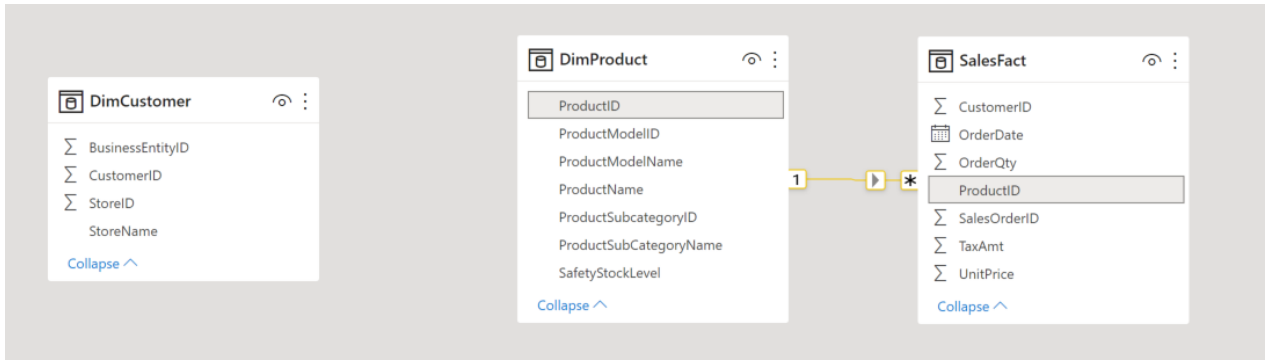
	ProductID	SalesOrderID	CustomerID	OrderQty	UnitPrice	OrderDate	TaxAmt
1	725	47415	29864	2	202.332	2012-07-31 00:00:00.000	266.7193
2	776	43659	29825	1	2024.994	2011-05-31 00:00:00.000	1971.5149
3	771	43875	29624	6	2039.994	2011-07-01 00:00:00.000	11871.5033
4	756	44093	29708	1	874.794	2011-08-01 00:00:00.000	1021.8926
5	757	44298	29824	1	874.794	2011-08-31 00:00:00.000	2656.7147
6	774	44514	30107	2	2039.994	2011-10-01 00:00:00.000	4095.822
7	750	44622	13771	1	3578.27	2011-10-09 00:00:00.000	286.2616
8	776	44794	29915	2	2024.994	2011-10-31 00:00:00.000	388.7988
9	749	45143	29278	1	3578.27	2011-12-12 00:00:00.000	286.2616
10	745	45329	29507	1	809.76	2012-01-01 00:00:00.000	5940.8882
11	729	45556	29746	2	183.9382	2012-01-29 00:00:00.000	2199.474

Using PowerBI to visualize our Start Schema

Start by opening up PowerBI. Go to get data > Azure > Azure Synapse Analytics. For the server, enter your dedicate sql endpoint server click okay. On the next screen select database and use your admin credentials. It would show your dedicated sql pool. Select your dimension and sales fact table and load.

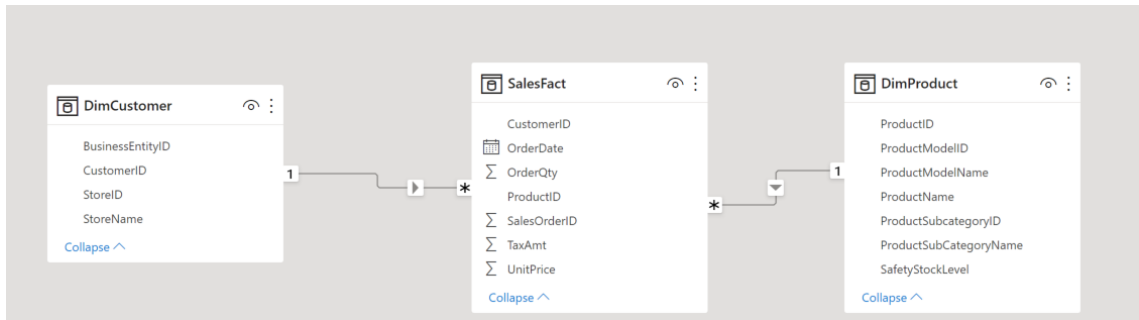


Once the data is loaded, go to data modelling from the left-hand menu. It has already recognized that there is some sort of connection between the DimProduct table and the SalesFact table.

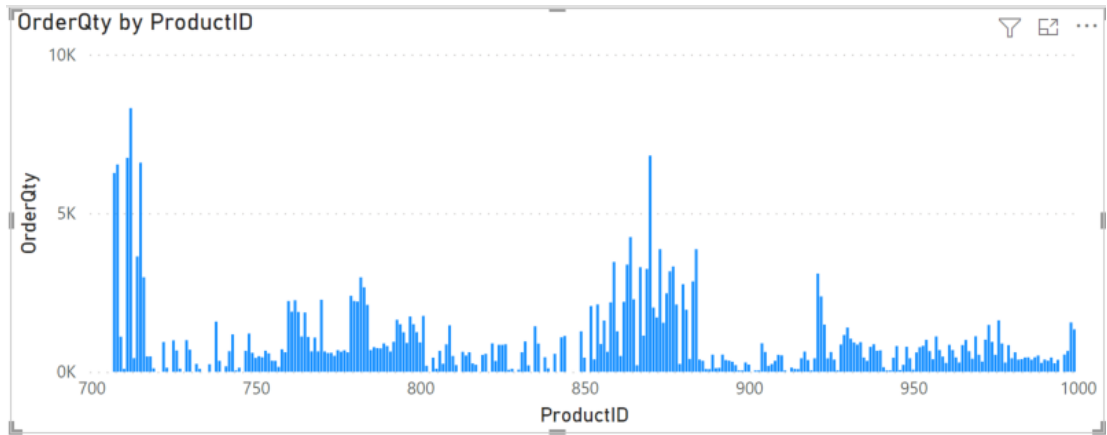


To create a relationship between the SalesFact table and the Customer table, right click on the Customer table and click new relationship. Select Customer table at the top and customerID column at the same time select SalesFact table with the customerID column and hit okay.

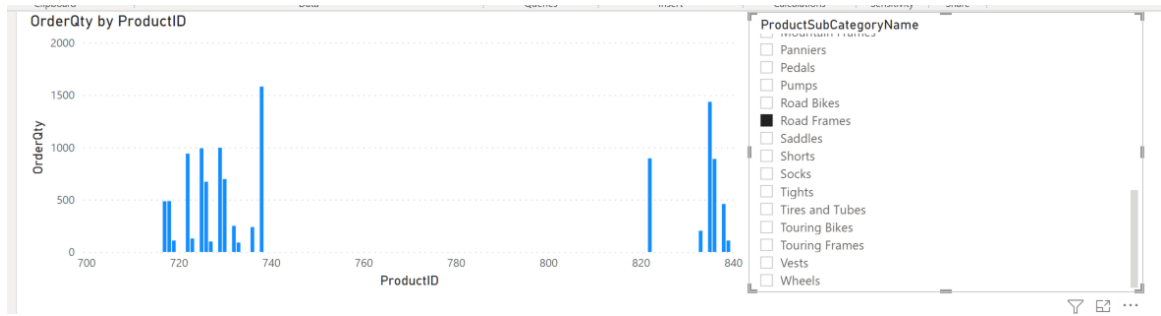
Now our model looks like this –



Now we can head back to the report screen and start creating visualizations based on the modelled data. We start by using the clustered column chart. For the axis we use product id from the salesfact table and values would be order quantity. We get –



Now we select a slicer and select product subcategory name. With it we get a list of product subcategories and upon selection a certain product sub category only the order quantity vs product id would be shown for that product subcategory. We get –



Understanding the Azure Synapse Architecture

There are many different types of tables that you can create in Azure Synapse in your dedicated SQL Pool. There are specifically three types of tables that are meant to help with the distribution of data in the dedicated sql pool.

1. Hash-distributed table
2. Replicated tables
3. Round-robin distributed tables

The reason why we have these different types of tables is so that there is efficiency since data warehouses would be processing a lot of data.

When ***we execute a query*** something, that query goes to the ***control node***. The control node ***distributes*** that query to different ***compute nodes*** which would take the query and process the query in parallel.

The number of compute nodes allocated to your dedicated sql pool is dependent on the number of data warehousing units allocated to your dedicated sql pool.

So the next layer after compute nodes is ***distributions***. These are just allocations of space where data is split across all these spaces. By default there would always be 60 distributions.

The above table mentioned depend on the architecture explained above.

Round-robin distributed tables

This is a table where data is evenly or as evenly as possible distributed among all the distributions without the use of a hash function. The assignment of rows to distributions is random. As a result the system sometimes needs to invoke a data movement operation to better organize your data before it can resolve a query. This extra step can slow down the queries.

We use the round robin table for the following scenarios -

- When getting started as a simple starting point since it is the default
- If there is no obvious joining key
- If there is no good candidate column for hash distributing the table
- If the table does not share a common join key with other tables
- If the join is less significant than other joins in the query
- When the table is a temporary staging table

By default all tables that are copied are in the form of round-robin distributed tables.

Hash Distributed tables

A hash distributed table distributes table rows across the compute nodes by using a deterministic hash function to assign each row to one distribution. Since identical values always hash to the same distribution, SQL analytics has built in knowledge of the row locations. In dedicated sql pool this knowledge is used to minimize the data queries which improves the query performance.

Some cases under which we can use the hash distributed table are:

- The table size on disk is more than 2 GB
- The table has frequent insert, update and delete operations

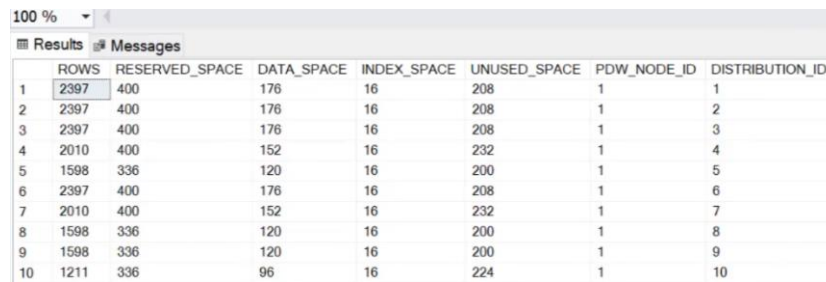
The column to choose for distribution should have evenly distributed unique values, should not have nulls, should not be a date column, is frequently used in join, group by, having clauses and is not used in the where clauses.

Replicated tables

Here a whole copy of the table is stored at each distribution.

To see what kind of distribution we have, run the command –

```
DBCC PDW_SHOWSPACEUSED('[dbo].[SalesFact]')
```



	ROWS	RESERVED_SPACE	DATA_SPACE	INDEX_SPACE	UNUSED_SPACE	PDW_NODE_ID	DISTRIBUTION_ID
1	2397	400	176	16	208	1	1
2	2397	400	176	16	208	1	2
3	2397	400	176	16	208	1	3
4	2010	400	152	16	232	1	4
5	1598	336	120	16	200	1	5
6	2397	400	176	16	208	1	6
7	2010	400	152	16	232	1	7
8	1598	336	120	16	200	1	8
9	1598	336	120	16	200	1	9
10	1211	336	96	16	224	1	10

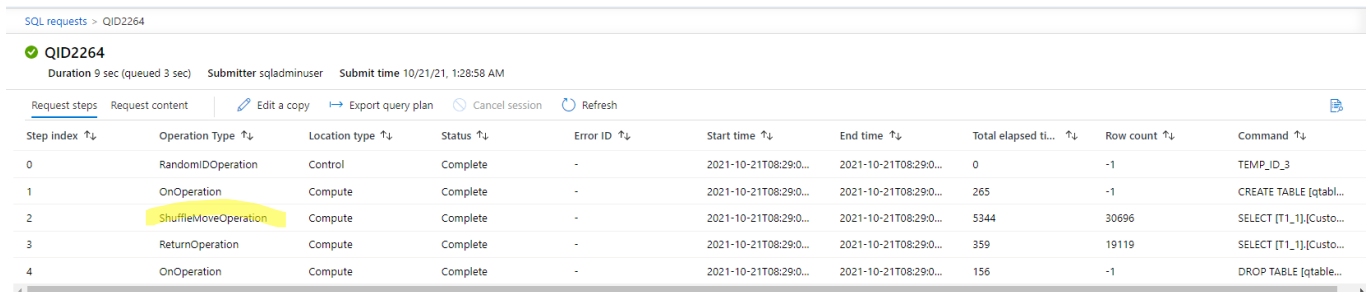
In the above picture we can see the distribution id is going from 1-60 and we just have one compute node.

A way to see how much work is being done while doing a particular operation (specifically a group by or join) on a round robin table, we can execute a group by query on a table

```
SELECT [CustomerID], COUNT([CustomerID]) as COUNT FROM [dbo].[SalesFact]
GROUP BY [CustomerID]
ORDER BY [CustomerID]
```

and then go to Monitor in our Synapse Studio and select SQL requests. We can select our database for pool and find our group by query.

Upon selecting the query we get something like this –



Step index	Operation Type	Location type	Status	Error ID	Start time	End time	Total elapsed ti...	Row count	Command
0	RandomIDOperation	Control	Complete	-	2021-10-21T08:29:0...	2021-10-21T08:29:0...	0	-1	TEMP_ID_3
1	OnOperation	Compute	Complete	-	2021-10-21T08:29:0...	2021-10-21T08:29:0...	265	-1	CREATE TABLE [qtabl...
2	ShuffleMoveOperation	Compute	Complete	-	2021-10-21T08:29:0...	2021-10-21T08:29:0...	5344	30696	SELECT [T1_1].[Custo...
3	ReturnOperation	Compute	Complete	-	2021-10-21T08:29:0...	2021-10-21T08:29:0...	359	19119	SELECT [T1_1].[Custo...
4	OnOperation	Compute	Complete	-	2021-10-21T08:29:0...	2021-10-21T08:29:0...	156	-1	DROP TABLE [qtable...

The shufflemoveoperation is the one that consumed the most time and also processed the most rows. This happened because the data is randomly distributed across the distributes and for the group by operation all the data had to be accessed.

Creating a Hash Distributed table

For this part we would be using [this](#) script.

We first drop the table. Then we create the table with the distribution given as Hash on CustomerID.

After creating the table, we would go to the Copy Data tool in the Azure Synapse Studio. We replicate our earlier pipeline.

- Source connection would be AdventureWorks with the table dbo.SalesFact
- Target connection would be AzureSynapseAnalytics and we select the target table as dbo.SalesFact
- Give the Synapse container in the AzureDataLakeStorage in the staging account linked service, in advanced select bulk insert

Once the pipeline is run. We can see the data has been loaded into the SalesFact table on the Synapse dedicated SQL pool.

When choosing a column as distribution column, avoid data and processing skew. Data skew means the data is not distributed evenly across the distribution. Processing skew means that some distributions take longer than others when running parallel jobs.

We again run the same group by common that was use earlier and go check the sql requests section

<div><div>QID2660</div><div>Duration 1 sec (queued 1 sec) Submitter sqladminuser Submit time 10/21/21, 2:00:14 AM</div></div>									
<div><div>Request steps</div><div>Request content</div><div><div>Edit a copy</div><div>Export query plan</div><div>Cancel session</div><div>Refresh</div></div></div>									
Step index	Operation Type	Location type	Status	Error ID	Start time	End time	Total elapsed ti...	Row count	Command
0	ReturnOperation	Compute	Complete	-	2021-10-21T09:00:1...	2021-10-21T09:00:1...	218	19119	SELECT [T1_1][Custo...

We see that there is only one return operation here compared to all the other operations that took place in round robin.

We should use *hash distributed* tables for our *fact table* and *replicated tables* for *dimension tables*. This can also be tested by creating the dimension tables as round-robin which they are by default and checking how the sql request looks like if we do a join between SalesFact (hash-distributed) and dimension table (round robin). So, when a join is executed all the distributed dimension table data is also shuffled to get all the data.

This shuffling step would not be present if all the data from the distributed table was present at every distribution as replicated tables.

[Design tables using Synapse SQL in Azure Synapse Analytics](#) is very informational.

Windowing functions in the dedicated sql pool

You are using [this](#) query for testing it out. I’m skipping this part as there are many online places where one can learn window functions.

Reading JSON files

We use the OPENROWSET function to read the json files.

We first upload the [log.json](#) file onto our container where Log.csv already exists.

Now we go to our Azure Synapse Studio, go to Data from the left, then Linked Data. Select your storage where this file is uploaded. The data lake would show up here is because we have done work with the container. Browse to the log.json file.

Right click on it and select the option “Select top 100 rows”. This would open up the editor with a script that’s already generated to read the top 100 rows from the json file.

Change the ROWTERMINATOR from 0x0b to 0x0a and then run the file.

```
1 -- this is auto-generated code
2 SELECT TOP 100
3     jsonContent
4 /* --> place the keys that you see in JSON documents in the WITH clause:
5     , JSON_VALUE (jsonContent, '$.key1') AS_header1
6     , JSON_VALUE (jsonContent, '$.key2') AS_header2
7 */
8 FROM
9     OPENROWSET(
10         BULK 'https://datalake203prep.dfs.core.windows.net/data/raw/log.json',
11         FORMAT = 'CSV',
12         FIELDQUOTE = '0x0b',
13         FIELDTERMINATOR = '0x0b',
14         ROWTERMINATOR = '0x0a'
15     )
16 WITH (
17     jsonContent varchar(MAX)
18 ) AS [result]
```

Results Messages

Table

Chart

Export results

Search

jsonContent

["id":35858,"Correlationid":"7104e9d9-944e-4195-a312-2240dcb9de9d","Operationname":"Delete VirtualNetworkGatewayConnection","Status":"Succeeded","Eventcategory":"Adminis...
["id":17934,"Correlationid":"541734f1-e2a7-441b-87e8-88b4a540e88e","Operationname":"Delete database accounts","Status":"Succeeded","Eventcategory":"Administrative","Level":"I...
["id":1,"Correlationid":"66641e13-d19f-4ce5-aafd-9d5d7bfa557","Operationname":"Delete SQL database","Status":"Succeeded","Eventcategory":"Administrative","Level":"Information...
["id":35859,"Correlationid":"7104e9d9-944e-4195-a312-2240dcb9de9d","Operationname":"Delete Virtual Machine","Status":"Succeeded","Eventcategory":"Administrative","Level":"Inf...

We get the above output. We are getting one row for the entire row of json data.

So to read it properly, we use [this](#) script. In this script we are mapping each row as json content. Then we are using JSON_VALUE function to take each property from the json content. Lastly, we are using cast to convert the data into the intended datatype.

We get a table as output –

```
1 SELECT
2     CAST(JSON_VALUE(jsonContent,'$.Id') AS INT) AS Id,
3     JSON_VALUE(jsonContent,'$.CorrelationId') AS Correlationid,
4     JSON_VALUE(jsonContent,'$.Operationname') AS Operationname,
5     JSON_VALUE(jsonContent,'$.Status') AS Status,
6     JSON_VALUE(jsonContent,'$.Eventcategory') AS Eventcategory,
7     JSON_VALUE(jsonContent,'$.Level') AS Level,
8     CAST(JSON_VALUE(jsonContent,'$.Time') AS datetimeoffset) AS Time,
9     JSON_VALUE(jsonContent,'$.Subscription') AS Subscription,
10    JSON_VALUE(jsonContent,'$.Eventinitiatedby') AS Eventinitiatedby,
11    JSON_VALUE(jsonContent,'$.Resourcetype') AS Resourcetype,
12    JSON_VALUE(jsonContent,'$.Resourcegroup') AS Resourcegroup
13 FROM
14     OPENROWSET(
15         BULK 'https://datalake203prep.dfs.core.windows.net/data/raw/log_json',
16         FORMAT = 'CSV',
17         FIELDQUOTE = '0x0b',
18         FIELDTERMINATOR = '0x0b',
19         ROWTERMINATOR = '0x0a'
20     )
21 WITH (
22     jsonContent varchar(MAX)
23 ) AS [rows]
```

Results Messages

View Table Chart Export results

Search

Id	Correlationid	Operationname	Status	Eventcategory	Level	Time	Subscription	Eventinitiatedby
35858	7104e9d9-944e...	Delete VirtualN...	Succeeded	Administrative	Informational	2021-03-22T19:...	20c6eec9-2d80...	techsup1000@...
35870	5aef1e41-7b6f...	Create or Upda...	Succeeded	Administrative	Informational	2021-03-22T17:...	20c6eec9-2d80...	techsup1000@...
35884	c70c021c-477b...	Create or Upda...	Started	Administrative	Informational	2021-03-22T18:...	20c6eec9-2d80...	techsup1000@...

Surrogate keys for Dimension Tables

We can use the identity column feature for this. We will refer to [this](#) script. Open up SSMS and drop the DimProduct table and then recreate it on your dedicated sql pool using –

```
CREATE TABLE [dbo].[DimProduct](
    [ProductSK] [int] IDENTITY(1,1) NOT NULL,
    [ProductID] [int] NOT NULL,
    [ProductModelID] [int] NOT NULL,
    [ProductSubcategoryID] [int] NOT NULL,
    [ProductName] varchar(50) NOT NULL,
    [SafetyStockLevel] [smallint] NOT NULL,
    [ProductModelName] varchar(50) NULL,
    [ProductSubCategoryName] varchar(50) NULL
)
```

Now we go to the copy data tool again to fill data into this table. We select source as AdventureWorks and select the DimProduct table. In the target select AzureSynapseAnalytics click next. In the next screen you’d see there is no source for the ProductSK column. We remove the column from column mapping as the values would be generated automatically.

Choose how source and destination columns are mapped

Table mappings (1)

Source

dbo.DimProduct

Target

dbo.DimProduct

Error

Please fix the errors

Column mappings

Type conversion settings

+ New mapping

Clear

Reset

Delete

Source	Type	Destination	Type
<div><div></div><div>Select or edit column</div></div>		ProductSK	int
<div>Mapping source is empty.</div>			
ProductID	int	ProductID	int

Select the same staging area, in advanced select bulk insert and create. Upon querying we see the ProductSK column –

	ProductSK	ProductID	ProductModelID	ProductSubcategoryID	ProductName	SafetyStockLevel	ProductModelName	ProductSubCategoryName
1	17	680	6	14	HL Road Frame - Black, 58	500	HL Road Frame	Road Frames
2	77	706	6	14	HL Road Frame - Red, 58	500	HL Road Frame	Road Frames
3	137	707	33	31	Sport-100 Helmet, Red	4	Sport-100	Helmets
4	197	708	33	31	Sport-100 Helmet, Black	4	Sport-100	Helmets
5	257	709	18	23	Mountain Bike Socks, M	4	Mountain Bike Socks	Socks
6	317	710	18	23	Mountain Bike Socks, L	4	Mountain Bike Socks	Socks

Changing Dimension tables

There are different kinds of changes that could occur on the dimension table, type1 , type2 and type 3 just like diabetes 😊

- Type1 : If there is an update in let's say the product name then we can just update the product name in the dimension table.
- Type2 : If there is an update in the name of the product but you also want to keep the old value and also add the new values. To facilitate this, we can introduce two additional columns. StartDate and EndDate and just fill them based on product name lifecycle. We can also have one more column IsCurrent which would be true for the current Product name.
- Type3 : IF there is an update on the product name then we would just have the original name and changed name column

Creating Heap tables

If we are creating a staging table and doing transformation on it and then loading it into another place, we can create a heap table for our staging data.

Heaps can be used as staging tables for large, unordered insert operations. Because data is inserted without enforcing a strict order, the insert operation is usually faster than the equivalent insert into a clustered index. If the heap's data will be read and processed into a final destination, it may be useful to create a narrow nonclustered index that covers the search predicate used by the read query.

We use the following script to create a heap table.

```
CREATE TABLE [dbo].[SalesFact_staging](
    [ProductID] [int] NOT NULL,
    [SalesOrderID] [int] NOT NULL,
    [CustomerID] [int] NOT NULL,
    [OrderQty] [smallint] NOT NULL,
    [UnitPrice] [money] NOT NULL,
    [OrderDate] [datetime] NULL,
    [TaxAmt] [money] NULL
)
WITH(HEAP,
DISTRIBUTION = ROUND_ROBIN
)
```

We create an index for ProductID

```
CREATE INDEX ProductIDIndex ON [dbo].[SalesFact_staging] (ProductID)
```

Syntax of how to use the **Case** statement –

```
SELECT [productid],[productname],
status = CASE [productstatus]
WHEN 'W' THEN 'Warehouse'
WHEN 'S' THEN 'Store'
WHEN 'T' THEN 'Transit'
END
FROM [ProductDetails]
```

Table Partitions

There are already distributions for different table types when it comes to our dedicated sql pools. We also have partitions, helps to divide rows of data into different logical partitions. These partitions help to divide the data into smaller groups of data.

Normally data is partitioned by dates. Also helps filtered data when using the where clause in your queries. Here the engine can just process the data in partition based on the condition mentioned in the where clause.

Creating a table with partitions

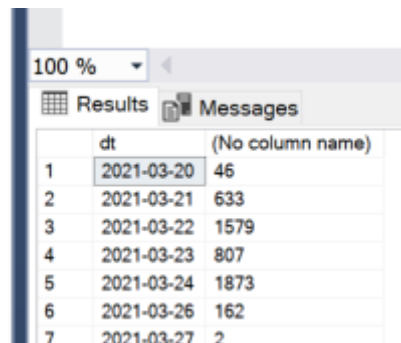
We would be using [this](#) file.

Head over to your dedicate sql pool through SSMS and drop your logdata table. Then create the logdata table without any partitions using the cleaned version of your Log.csv file

Using the following query we see the distribution of data based on the dates

```
SELECT FORMAT(Time,'yyyy-MM-dd') AS dt,COUNT(*) FROM logdata
GROUP BY FORMAT(Time,'yyyy-MM-dd')
```

We get something like –



	dt	(No column name)
1	2021-03-20	46
2	2021-03-21	633
3	2021-03-22	1579
4	2021-03-23	807
5	2021-03-24	1873
6	2021-03-26	162
7	2021-03-27	2

We observe that there are only a few months present in the data.

Drop the table again and recreate it using the date partitions.

We are using right partitions here –

```
PARTITION ( [Time] RANGE RIGHT FOR VALUES
('2021-04-01','2021-05-01','2021-06-01')
```

The above query would put all the dates before the first of each month in a separate partition. Had the query been range left, then all the data after the first of each month would be in a separate partition.

You won't see any difference on the results when you do a select * statement since the way the data is being stored is changed. There is Microsoft documentation link s which would show the different partitions that there are for our table.

General practice with partitions is to not have too many partitions. You have to remember that your data will be distributed across distribution and partitions. For optimal compression and performance of clustered columnstore tables, a minimum of 1 millions rows per distribution and partition are needed.

Advantages of Partitions

If we use the time column in a where clause then only the partitions that fall under the where clause would be accessed hence it is more efficient since it is not doing an entire table scan.

Also, we can move partitions around. We can create a new table with specific partitions and switch the partition from an existing table to the newer table.

```
CREATE TABLE [logdata_new]
WITH
(
DISTRIBUTION = ROUND_ROBIN,
PARTITION ( [Time] RANGE RIGHT FOR VALUES
            ('2021-05-01','2021-06-01')

) )
AS
SELECT *
FROM logdata
WHERE 1=2

-- Switch the partitions and then see the data

ALTER TABLE [logdata] SWITCH PARTITION 2 TO [logdata_new] PARTITION 1;
```

In the newer table there's only data for April which would be in partition 2.

el	Time	Subscription
rmational	2021-04-30 18:11:23.000	20c6eec9-2d80-
rmational	2021-04-26 19:03:47.000	20c6eec9-2d80-
rmational	2021-04-15 06:12:15.000	20c6eec9-2d80-
rmational	2021-04-30 18:15:48.000	20c6eec9-2d80-
rmational	2021-04-26 19:45:08.000	20c6eec9-2d80-
rmational	2021-04-15 06:21:21.000	20c6eec9-2d80-

So the original table now only has three partitions and the newer table has partition 2 of the original table.

Indices (Indexes ?)

Columnstore

Tables are by default created as clustered columnstore index tables. So data warehousing systems are generally stored column by column on the infrastructure. This makes it faster to working with the data. Columnstore index providves the highest level of data compression and the best overall query performance.

They are not ideal when you have max amount of data in the columns in the from of varchar, nvarchar or varbinary.

Also these are note idea for small tables that have less than 60 million rows.

Clustered index

Can create a clustered index on just a specific column of a table.

Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition. There can be only one clustered index per table, because the data rows themselves can be stored in only one order.

If there are a few lookups which make use of a very specific column you can consider creating a clustered index. Clustered indexes may outperform clustered columnstore tables when a single row needs to be quickly retrieved.

NonClustered index

Nonclustered indexes have a structure separate from the data rows. A nonclustered index contains the nonclustered index key values and each key value entry has a pointer to the data row that contains the key value. This adds to the table space as we have a separate index along with the data. Clustered index is inbuilt into the table.