

CREATING AZURE SQL DATABASE:

Go to menu → SQL Data base → click on create button → enter subscription name and resource group to be selected and enter SQL data base name.

Now enter server name → if server is not available, click on create button → enter server name,

- Enter location,
 - Enter Authentication (1. Use Azure AD authentication, 2. Use both SQL Authentication and Azure AD Authentication, 3. Use SQL Authentication)
 - Server admin login: xxxxxx(can be sql admin like that)
 - Password:
 - Confirm Password:
- Click on Ok.
Now server is created.

In Compute+Storage details → click on Configure database

We will select service tier:

Compute tier: Basic/standard/high scalability/business needs for high availability and performance/General

Click on next → in networking tab: we will configure network access and connectivity for our server.

Connectivity method: (No access, public endpoint, Private endpoint)

Fire wall rules:

Allow azure services and resources to access this server: click on Yes/no

Add current client IP address: Yes/No

Connection policy: configure how clients communicate with our SQL data base server.
(Default,proxy,redirect)

Click on next → Security

Click on next → Additional Settings → use existing data as (None/Backup/Sample database)

Click on next tags → review and create → create

CREATION OF AZURE STORAGE ACCOUNT:

Enter subscription name, resource group name,

Instance details: Storage account name, Region, Performance (Standard/Premium), Redundancy (LRS, GRS, ZRS, GZRS).

Redundancy means make read access to data available in the event of regional unavailability.

Locally redundant Storage (LRS): Recommended for non-critical scenarios. Protection against server rack and drive failures in single data center.

Geo Redundant Storage (GRS): Recommended for backup scenarios. If in 1 region 3 data center there means, 1 data center failed, we can have access to other 2 data center data availability as backup

Zone Redundant Storage (ZRS): Recommended for high availability scenarios. Protection against data center level failures.

GZRS: Recommended for critical data scenarios.

Click on next, in advanced, enable hierarchical namespace. Click on create.

In ADLS, we can store semi structured data (csv json, parquet) in containers.

Structured data: Table with rows and columns

Semi-structured data: CSV, JSON, PARQUET

Unstructured data: Videos, Audios

Azure data lake gen2 is one place storage of all our data (Structured, Semistructured, Unstructured) that can be analyzed at later point of time.

Azure SQL Database: has structured data with them -> It is an OLTP system used to record day to day operations.

DATA WAREHOUSE: We can host a SQL Data warehouse via Azure Synapse Service. Data warehouse is a centralized repository of data taken from different data sources. DWH used by business for analyzing data. Senior management of company want to perform some analysis on data where in all these records are being hosted like Business want to understand how they are performing on a year-to-year basis like how sales is this year when compared to previous year. This sort of analysis is based on history, historical data. If we try to perform these historical data (past year) queries on the data base that is actually used for our day-to-day transactions (Azure SQL Database), it might actually put a strain on the data base itself. If you have your business users also creating reports, performing analysis on the data, at the same time, it could put a pressure on the database system.

So, single data base system can't be used for day-to-day transactions and for analysis. So, in this case, what is normally done is, you take your data that's required from your O L T P system and transfer it onto something known has a SQL data warehouse (called as OLAP -Online Analytical Processing System)

Now for SQL Data warehouse, we need some tool, some sort of engine to drive SQL data warehouse. Data warehouse used to store large amounts of information of structured data only. SQL DWH can store data of past years (up to 5 years /10 years). This gives capability for business users to see or to analyze the data over time. So, the data can be analyzed accordingly in the data warehouse. in order to analyze so much of data, you'd have a lot of processing power in place. when it comes onto analyzing a lot of data because see, you might be having a million or even a billion rows within a SQL data warehouse. In order to analyze and process the data you need to have a lot of compute power in place. Just like the SQL database, you can fire SQL queries against SQL data warehouse. Business users can use reporting tools to visualize the data in the warehouse itself.

We won't copy data directly from SQL data base to SQL DWH for analysis. we should decide what data we need to take; it may not be all columns data. You might want to only take certain columns from a table in terms of the data and then transfer it onto a data warehouse. You might also need to perform operations or just cleaning your data. In our SQL data base, if we have rows which had null values, we don't want those rows of null values. You want to clean your data, you want to take only those rows which have values onto the SQL data warehouse. You might need to also perform transformations on some of the columns of data. So that's up to us based on business needs.

When we are taking data from source system (Azure SQL Database), we perform some transformations before putting it in to SQL Data warehouse. When it comes on to videos, we will basically take information about videos from ADLS and probably transform it on to tables in a SQL Data warehouse and then we could build reports and visualize our data based on the data in the data warehouse itself.

CREATION OF AZURE SYNAPSE:

All Resources → Create a Resource → search synapse and enter → create →
Enter subscription, Resource group.

Workspace details:

We will enter unique workspace name, select a location, and choose a Primary Data Lake Storage Gen2 file system to serve as a default location for logs and job output and click on check box of assign myself the Storage Blob data contributor role on the Data Lake Storage Gen2 account to interactively query it in the workspace. Click on next tab.

In Networking tab, enter authentication method for access to workspace (Local and Azure AD authentication/ use only Azure AD authentication). enter SQL Server admin login and password, go to next review, and create.

Use of Synapse workspace is to analyze the data that we have in our data lake.

There are 2 compute options in azure synapse 1.) Serverless SQL Pool 2.) Dedicated SQL Pool

Dedicated SQL Pool: is used to build our data warehouse. If we want to have tables in place with data, if we need to persist (remain/store/keep on going/ఎల్లకాలం ఉండు) our data, we can use SQL Pool option. Here we will have separate computing nodes that can be used for processing our data. Here we will be charged based on metric known as DWU (Data warehousing units). This option is more expensive when it comes on to Azure Synapse. but here your data is already structured, ready to be queried and visualized by your business users. See, your business users don't want to have the job of removing null values, cleaning data, etc. They want to have clean structured data in place so they can start directly performing analysis of the data. The first job that could be done by data engineers is to cleanse your data, transform the data, and then put in a SQL data warehouse to be analyzed by the business users. So, your business users will probably be working with the data in the SQL pool, in the dedicated SQL pool in your SQL data warehouse.

Serverless SQL Pool: The other compute option we have is serverless SQL Pool i.e.; no servers involved here. All of the compute infrastructure that is used for analyzing our data in the background is managed by the service itself. Here we only get charged for the amount of data that we analyze in the SQL Pool.

In dedicated SQL Pool, we get charged based on DWU, compute and Storage. But in the serverless pool, we only get charged based on how much of data that we analyze. Serverless SQL Pool is used to analyze the data within our Azure Data Lake Gen2 Storage Account.


We can use serverless SQL pool to quickly perform ad-hoc analysis of our data. Let's say in Azure storage account, in Azure Data Lake Gen2 storage account, we can initially use serverless SQL Pool option to analyze our data and then you could use the tools within Azure Synapse to transfer data onto the dedicated SQL pool. Here, you are charge based on how much you use. In the serverless SQL pool, you cannot persist data.

The most important difference between dedicated SQL Pool and Serverless SQL pool is when we want to host SQL Data warehouse where in we have persisted data that is stored, we will use dedicated SQL pool whereas if we want to perform quick analysis of data let's say in Azure data lake gen2 storage account, we can make use of serverless SQL Pool. Data is not persisted in serverless SQL Pool.

When we open azure synapse studio, we can see sections at left hand side.

Data Section: we can have workspace storage account that we created during creation of azure synapse workspace, and data file system or data container is also available in linked tab.

If I want to bring data from data lake gen2 storage to workspace, we need to go to Data section → click

on workspace tab, click on , click connect to external data, we will choose Azure Data Lake Storage Gen2, continue button →

Now we will create new linked service here (we are linking our workspace on to Azure Data Lake Gen2 Storage Account)

Name: Azure Data Lake Storage Gen2 account name

Authentication type: Account key

Subscription:

Storage Account name: ADLS Gen2 account name

Test Connection: To Linked Service.

Click on create.

Now if we go to Data → Linked Service section → now we can see data lake storage account attached here.

Right click on csv file → NEW SQL SCRIPT → Select TOP 100 rows →

Using Azure Synapse, we can execute typical SQL query language statements against a CSV file located in our Azure Data Lake Gen2 Storage account.

When we query against csv file which is present in ADLS Gen2 with serverless SQL Pool, we will get file doesn't exist or renamed. If we get so we need to authorize ourselves to be able to access the data within our storage account.

So, for that go to Storage Account → Access Control (IAM) → Add Role Assignment -> select Storage Blob Data Reader, next → select members (mail id) → next and review and assign. I want to be able to read only data within my Data Lake Gen2 Storage Account, so Storage Blob Data Reader role is chosen.

Develop Section: Here we can create SQL Scripts, KQL Scripts, Notebook, Dataflows, Apache Spark Job definition.

Integrate Section: we can build our pipelines.

EXTERNAL TABLES: An external table can be used to read or write data in Hadoop, in Azure Blob Storage, or Azure Data Lake Storage. External tables are possible in both your serverless SQL pool and your dedicated SQL pool.

OPENROWSET function allows us to view the data. But with the help of external tables, we can now define the structure of a table like we would do in normal relational database.

For using external table 3 commands required are:

CREATE EXTERNAL DATA SOURCE: This is used to specify external storage that needs to be referenced. What is the location of our data source itself we will specify here.

CREATE EXTERNAL FILE FORMAT: What is the external file format we are using CSV, Parquet

CREATE EXTERNAL TABLE: It will create the table definition.

Master database in a Microsoft SQL server-based engine contains system base tables. If we want to have our tables defined within a user-defined database, we can first issue the create database command to create a database itself.

```
CREATE DATABASE [appdb]
```

```
CREATE EXTERNAL DATA SOURCE data_source_name  
WITH ( LOCATION= 'https://datalake2000233.dfs.core.windows.net/csv'  
      CREDENTIAL = SasToken)
```

From above create query I only need to give location path till our container name in LOCATION . Now, along with specifying our location we have to specify what method we are gonna use to actually authenticate or basically authorize this script to access the particular container. So, Credential = Sas Token we mentioned. We have to define SasToken here.

For authenticating external data source, we will see credential = SasToken, this credential will allow the script to actually connect or to access the data within a Azure Data Lake Gen2 Storage Account.

<https://learn.microsoft.com/en-us/sql/t-sql/statements/create-external-data-source-transact-sql?view=azure-sqldw-latest&tabs=dedicated>

This link will have data of external data source and its path with prefix information.

For external data source as Azure Data Lake Storage Gen2 , there are 3 ways we can access our container. 1 is abfs[s](Azure blob file system driver) 2 is http[s] 3. Wasb[s] (Windows Azure Storage driver). We mostly use http[s] for accessing our container.

CREATE DATABASE SCOPED CREDENTIAL : This allow us to create kind of password.

Syntax:

```
CREATE DATABASE SCOPED CREDENTIAL credential_name  
WITH IDENTITY ='Identity_name' , SECRET='secret_value';
```

IDENTITY Command is used to specify that we want to use shared access signature method.

Secret command is used to get secret key from Azure Data Lake Storage account .

```
CREATE DATABASE SCOPED CREDENTIAL SasToken
WITH IDENTITY = 'SHARED ACCESS SIGNATURE' ,
SECRET = '' ; here we will enter generated sas token.
```

When we execute `CREATE DATABASE SCOPED CREDENTIAL` above command, it will give us an error saying that there is no master encryption key

Master encryption key is used to encrypt Shared Access Signature. Shared Access Signature (Generated SAS Token is now defined in our data base created (appdb) . This appdb database will now use shared access signature to access now the container in our Data Lake Gen2 storage account.

`CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'P@ssw0rd@123';` This key is used to protect Shared Access Signature. database master key is used to protect other keys, certificates, and secrets within the database.

We need to run now commands in below order.

-- First we need to create a database in the serverless pool

```
CREATE DATABASE [appdb]
```

-- Ensure to switch the context to the new database first from default data base.

-- Here we are creating a database master key. This key will be used to protect the Shared Access Signature which is specified in the next step

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'P@ssw0rd@123';
```

-- Here we are using the Shared Access Signature method to authorize the use of the Azure Data Lake Storage account

```
CREATE DATABASE SCOPED CREDENTIAL SasToken
```

```
WITH IDENTITY='SHARED ACCESS SIGNATURE',
```

```
SECRET = 'sv=2021-12-02&ss=b&srt=sco&sp=rwl&se=2023-04-26T21:14:21Z&st=2023-04-26T13:14:21Z&spr=https&sig=rqBhNq4McZIZUMY2QP2mDnzZWgb9eZt1YTViVtVjWIo%3D';
```

-- This defines the source of the data.

```
CREATE EXTERNAL DATA SOURCE log_data
```

```
WITH (LOCATION = 'https://datalake2000233.dfs.core.windows.net/csv',
```

```
        CREDENTIAL = SasToken
```

```
)
```

CREATE EXTERNAL FILE FORMAT: This command will tell us what is the format of the file we are trying to access.

<https://learn.microsoft.com/en-us/sql/t-sql/statements/create-external-file-format-transact-sql?view=azure-sqldw-latest&tabs=delta>

if we go to this link , there will be file formats which will be supported to create external table.

```
CREATE EXTERNAL FILE FORMAT File_format_name WITH (
FORMAT_TYPE= DELIMITEDTEXT,
FORMAT_OPTIONS(
FIELD_TERMINATOR = ',' ,
FIRST_ROW = 2))
```

```
CREATE EXTERNAL FILE FORMAT TextFileFormat WITH (
FORMAT_TYPE= DELIMITEDTEXT,
FORMAT_OPTIONS(
FIELD_TERMINATOR = ',' ,
FIRST_ROW = 2))
```

Format_type will tell us what is a type of file that we are trying to access DELIMITED TEXT,JSON,PARQUET etc.

Format_Options tells us basically how to read the file.

```
CREATE EXTERNAL FILE FORMAT MyDelimitedTextFormat
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS (
        FIELD_TERMINATOR = '|',
        STRING_DELIMITER = '"',
        FIRST_ROW = 2
    )
);
```

Here with FIRST_ROW, we are saying please skip the first row because this contains header information so start reading data from 2nd row onwards.

--- Here we define external table

```
CREATE EXTERNAL TABLE [logdata]
(
    [Correlation id] [varchar](200) NULL,
    [Operation name] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Event category] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
    [Event initiated by] [varchar](1000) NULL,
    [Resource type] [varchar](1000) NULL,
    [Resource group] [varchar](1000) NULL,
```

```

[Resource] [varchar](2000) NULL)
WITH (
  LOCATION = '/Log.csv',
  DATA_SOURCE = log_data,
  FILE_FORMAT = TextFileFormat
)

```

We have enclosed [] braces in column names to ensure that we can also have the space between the column names.

We have defined structure of external table with in our serverless SQL Pool. Our data will not be in serverless SQL Pool. Our data is not in Azure Synapse. Our data is in the Azure Data Lake Gen2 Storage account. For SQL based Data base table structure and data is within the SQL Database. In this case, in Azure Synapse in the serverless SQL Pool, our table is defined within Azure Synapse, whereas the data is still in our Azure Data Lake Gen2 Storage Account.

If I query data of external table, select * from [logdata] , we can get below error .
 External table 'logdata' is not accessible because location does not exist, or it is used by another process.
 If we get so, which means that Shared Access Signature token may be expired or location of file in ADLS Gen2 may be incorrect. If SAS Token expired, then regenerate new and keep it in command of CREATE DATABASE SCOPED CREDENTIAL.

If we get this error while querying data, The maximum reject threshold is reached error which means column header in ADLS Gen2 file not matching with external table columns headers sometimes space issue between column headers also error (The maximum reject threshold is reached) can be thrown.

If I want to drop DATABASE SCOPED CREDENTIAL, we need to drop in below order only. otherwise, we will get error as can't drop a table due to dependency of other create commands.

```

DROP EXTERNAL TABLE [logdata]
DROP EXTERNAL DATA SOURCE log_data
DROP DATABASE SCOPED CREDENTIAL SasToken

```

```

CREATE DATABASE SCOPED CREDENTIAL SasToken
WITH IDENTITY='SHARED ACCESS SIGNATURE'
, SECRET = 'sv=2021-12-02&ss=b&srt=sco&sp=rwla&se=2023-05-30T21:37:55Z&st=2023-05-02T13:37:55Z&spr=https&sig=tYzY6clu1rEITvnw700MLApKzF3F1b3ChXd49vYa5z0%3D'

```

```

CREATE EXTERNAL DATA SOURCE log_data_parquet
WITH ( LOCATION = 'https://datalake2000233.dfs.core.windows.net/parquet',
  CREDENTIAL = SasToken
)

```

```

CREATE EXTERNAL FILE FORMAT parquetfile
WITH (
  FORMAT_TYPE = PARQUET,
  DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
);

```


Used DATA_COMPRESSION = '' here to generate the parquet-based file.

2 types of data compression techniques we can use any of technique here 1 is 'org.apache.hadoop.io.compress.SnappyCodec' → Snappy compression and 2 is 'org.apache.hadoop.io.compress.GzipCodec' --. Gzip compression, these 2 compression techniques are used to compress the file.

-- Here we define the external table

```
CREATE EXTERNAL TABLE [logdata_parquet]
(
  [Correlation id] [varchar](200) NULL,
  [Operation name] [varchar](200) NULL,
  [Status] [varchar](100) NULL,
  [Event category] [varchar](100) NULL,
  [Level] [varchar](100) NULL,
  [Time] [datetime] NULL,
  [Subscription] [varchar](200) NULL,
  [Event initiated by] [varchar](1000) NULL,
  [Resource type] [varchar](1000) NULL,
  [Resource group] [varchar](1000) NULL,
  [Resource] [varchar](2000) NULL)
WITH (
  LOCATION = '/log.parquet',
  DATA_SOURCE = log_data_parquet,
  FILE_FORMAT = parquetfile
)
Table will be created successfully.
SELECT * FROM [logdata_parquet]
```

-- Then we will get the error (this data from select query coming from parquet file of ADLS, external table has time data type DATETIME, parquet file in ADLS has column data type string so error will be thrown.

-- Column 'Time' of type 'DATETIME' is not compatible with external data type 'Parquet physical type: BYTE_ARRAY, logical type: UTF8', please try with 'VARCHAR(8000)'. File/External table name: 'logdata_parquet'.

When we open csv file 1st row is a header and will have all data with no data type mention, But in parquet based file, the structure of file is in such a way that the beginning of the file has, what is the column names along with the what is the data type for each column is mentioned. The benefit of having a data type with a column of information is you can actually execute specific commands on the column of data based on the data type. For example, in a SQL table, let's say that you have a column of order quantity. So all of that will be numbers, right? You will have two orders being placed, five orders being placed for one row, 10 orders for another, etc, etc. Know what you can do in your SQL-based language. You can use the sum aggregate function to find the total of the order quantity. That's because you know that this is of the type of the SQL engine, knows that this is of the type in integer, it's numbers. So, you can actually perform a summation of those numbers. If you had string, you could not do that. You could probably use string-based functions to work with string-based values. So that's the benefit of having a type associated with your column of data. So, within the CSV file, there is no association.

In order to change data type for column time, we need to drop external table now.

```
DROP EXTERNAL TABLE [logdata_parquet]
```

```
CREATE EXTERNAL TABLE [logdata_parquet]
(
  [Correlation id] [varchar](200) NULL,
  [Operation name] [varchar](200) NULL,
  [Status] [varchar](100) NULL,
  [Event category] [varchar](100) NULL,
  [Level] [varchar](100) NULL,
  [Time] [varchar] NULL,
  [Subscription] [varchar](200) NULL,
  [Event initiated by] [varchar](1000) NULL,
  [Resource type] [varchar](1000) NULL,
  [Resource group] [varchar](1000) NULL,
  [Resource] [varchar](2000) NULL)
WITH (
  LOCATION = '/log.parquet',
  DATA_SOURCE = log_data_parquet,
  FILE_FORMAT = parquetfile
)
```

Select * from [logdata_parquet] --- if we run select query now, results will be fetched but there will be nulls data for few columns . null data will be only coming for columns that have space between column names . So we need to remove spaces for column names as shown below.

```
CREATE EXTERNAL TABLE [logdata_parquet]
(
  [Correlationid] [varchar](200) NULL,
  [Operationname] [varchar](200) NULL,
  [Status] [varchar](100) NULL,
  [Eventcategory] [varchar](100) NULL,
  [Level] [varchar](100) NULL,
  [Time] [varchar] NULL,
  [Subscription] [varchar](200) NULL,
  [Eventinitiatedby] [varchar](1000) NULL,
  [Resourcetype] [varchar](1000) NULL,
  [Resourcegroup] [varchar](1000) NULL,
  [Resource] [varchar](2000) NULL)
WITH (
  LOCATION = '/log.parquet',
  DATA_SOURCE = log_data_parquet,
  FILE_FORMAT = parquetfile
)
```

Select * from [logdata_parquet] -- now results will be fetched .

In Parquet based file, by default, we should not be having spaces between column names.

When we are converting CSV file to parquet based file using ADF , we need to ensure to remove spaces in column names. Hence when we are creating external table, we have to remove spaces between column names.

We can read data from multiple files of csv or parquet within a location.

```
DROP EXTERNAL TABLE [logdata_parquet]
```

```
CREATE EXTERNAL TABLE [logdata_parquet]
(
    [Correlation id] [varchar](200) NULL,
    [Operation name] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Event category] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
    [Event initiated by] [varchar](1000) NULL,
    [Resource type] [varchar](1000) NULL,
    [Resource group] [varchar](1000) NULL,
    [Resource] [varchar](2000) NULL)
WITH (
    LOCATION = '/',
    DATA_SOURCE = log_data_parquet,
    FILE_FORMAT = parquetfile
)
```

For example, we deleted Log.parquet file from ADLS Gen2 account and if we query now, select * from [logdata_parquet]; we will get error as External table [logdata_parquet] is not accessible because location doesnot exist or it is used by another process. We got this error because we deleted file in ADLS when we query select query, it is taking data with in our Data lake file.

OPENROWSET FUNCTION: OPENROWSET along with BULK FUNCTION allows us to access files in the azure storage account. It actually gives functionality of quickly reading the contents of a remote data source file. We have seen that we could use the options of first creating an external data source, creating an external file format, then creating an external table. This kind of a longer way of reading data from files from our serverless SQL pool. But with those options, you get more flexibility on the type of files that you can access. At the same time, you have different security mechanisms that you can specify, and you can define your external table so that you can start working with a table accordingly. If you want just a quick way of reading data from your files in your Azure storage account, you can make use of the OPENROWSET function.

```
SELECT *
FROM OPENROWSET (BULK 'https://datalake2000233.dfs.core.windows.net/parquet/Log.parquet',
    FORMAT = 'PARQUET') AS [logdata_temp]
In this query reading data from only 1 file within
the container, URL we will get from file URL link where we will copy URL from Log.parquet file.
```

```
SELECT *
FROM OPENROWSET(BULK 'https://datalake2000233.dfs.core.windows.net/parquet/*.parquet',
    FORMAT = 'PARQUET') AS [logdata_temp]
```

This above select query will read data from all files of parquet container, so we kept *.parquet

For openrowset function, we are not specifying any sort of secret credentials because for ADLS gen2 storage account we already added IAM (Access Control) role assignment as Storage BLOB DATA READER role for my account.

DEDICATED SQL POOL/SQL POOL: Using dedicated SQL Pool, we can now build tables in our SQL Data warehouse. **with the serverless SQL pool, we could only build external tables wherein the structure of the table is defined in Azure Synapse whereas the data itself resides in an Azure storage account. But if you wanna persist data within Azure Synapse itself, you know this makes it much more efficient when you're working with large data tables and if you wanna perform analysis across those tables, then you should build a SQL data warehouse with the help of dedicated SQL Pool.**

CREATION OF DEDICATED SQL POOL: Open Synapse workspace → Left side go to analytic pools → click on SQL Pools → click on +, now dedicated SQL Pool creation wizard will come. This is like creating a data base within Azure Synapse itself. For the data base we will enter below details in wizard.

Dedicated SQL Pool name: poolddb (for the data base, we will give name poolddb/appdb whatever we like).

Performance level: choose performance level next → review and create. → create.

Now in synapse studio, in Data section → Workspace section we can see our created dedicated SQL pool poolddb database.

External tables work with both serverless SQL Pool and dedicated SQL Pool.

In serverless SQL pool, we can only create external tables. we cannot create normal tables. where as

In dedicated SQL pool, we can define tables and external tables.

CREATION OF EXTERNAL TABLES IN DEDICATED SQL POOL:

Dedicated SQL Pool means data base only. So, for the data base, we need to create master key encryption with password to protect secrets and keys.

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'P@ssw0rd@123';
```

-- Here we are using the Storage account key for authorization

```
CREATE DATABASE SCOPED CREDENTIAL SasToken
```

```
WITH IDENTITY='SHARED ACCESS SIGNATURE'
```

```
, SECRET = 'sv=2021-12-02&ss=b&srt=sco&sp=rlw&se=2023-04-26T21:14:21Z&st=2023-04-26T13:14:21Z&spr=https&sig=rqBhNq4McZIZUMY2QP2mDnzZWgb9eZt1YTViVtVjWIo%3D';
```

-- When it comes to Dedicated SQL pool

-- <https://learn.microsoft.com/en-us/azure/synapse-analytics/sql/create-use-external-tables>
-- For CSV files, we need to use the Hadoop driver in the TYPE when mentioning the data source
-- The native driver is supported for Parquet based files, so not mentioning TYPE = Hadoop in parquet file external data source creation.

```
CREATE EXTERNAL DATA SOURCE log_data_parquet
WITH ( LOCATION = 'https://datalake2000233.dfs.core.windows.net/parquet',
        CREDENTIAL = SasToken
)
```

```
CREATE EXTERNAL FILE FORMAT parquetfile
WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io. compress. SnappyCodec'
);
```

```
CREATE EXTERNAL TABLE [logdata_parquet]
(
    [Correlationid] [varchar](200) NULL,
    [Operationname] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Eventcategory] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [varchar](500) NULL,
    [Subscription] [varchar](200) NULL,
    [Eventinitiatedby] [varchar](1000) NULL,
    [Resourcetype] [varchar](1000) NULL,
    [Resourcegroup] [varchar](1000) NULL,
    [Resource] [varchar](2000) NULL
)
WITH (
    LOCATION = '/log.parquet',
    DATA_SOURCE = log_data_parquet,
    FILE_FORMAT = parquetfile
)
SELECT * FROM logdata_parquet
```

CREATION OF EXTERNAL TABLE USING CSV -DEDICATED SQL POOL:

```
CREATE DATABASE SCOPED CREDENTIAL AzureStorageCredential
WITH
    IDENTITY = 'datalake2000233',
```

```
SECRET =  
'ZuTNLYAYPx1gmjZHE2+plizxi4zwOHBYX395HDVQ/wM47yfhHBEBppzMKVegfROli99SCwCWD04F+AStQj  
vUxw==';
```

In the context of creating an external data source in Azure Synapse Analytics, specifying **TYPE = HADOOP** is valid. This is used when you want to create an external data source that connects to data stored in the Hadoop Distributed File System (HDFS) or Hadoop-compatible storage, such as Azure Data Lake Storage Gen1 or Gen2.

```
CREATE EXTERNAL DATA SOURCE log_data_csv  
WITH ( LOCATION = 'abfss://csv@datalake2000233.dfs.core.windows.net',  
        CREDENTIAL = AzureStorageCredential,  
        TYPE = HADOOP  
) -- we have change abfss here because we are using now Hadoop driver to read data in our csv based  
file so because of this we are now changing protocol to abfss.
```

```
CREATE EXTERNAL FILE FORMAT TextFileFormat WITH (  
    FORMAT_TYPE = DELIMITEDTEXT,  
    FORMAT_OPTIONS (  
        FIELD_TERMINATOR = ',',  
        FIRST_ROW = 2))
```

```
CREATE EXTERNAL TABLE [logdata]  
(  
    [Correlation id] [varchar](200) NULL,  
    [Operation name] [varchar](200) NULL,  
    [Status] [varchar](100) NULL,  
    [Event category] [varchar](100) NULL,  
    [Level] [varchar](100) NULL,  
    [Time] [datetime] NULL,  
    [Subscription] [varchar](200) NULL,  
    [Event initiated by] [varchar](1000) NULL,  
    [Resource type] [varchar](1000) NULL,  
    [Resource group] [varchar](1000) NULL,  
    [Resource] [varchar](2000) NULL  
)  
WITH (  
    LOCATION = '/Log.csv',  
    DATA_SOURCE = log_data_csv,  
    FILE_FORMAT = TextFileFormat  
)  
SELECT * FROM [logdata]
```

When we ran select query, we will get below error.

```
-- HdfsBridge::recordReaderFillBuffer - Unexpected error encountered filling record reader buffer:  
HadoopSQLException: Error converting data type VARCHAR to DATETIME.
```

Using hadoop driver, it is not able to convert data that we have in our csv file on to DATETIME here in TIME column as shown above. When we are using our native drivers in the serverless sql pool , it was fine. Conversion was possible and we are getting our data. When it comes onto hadoop driver,we are not able to convert data on to datetime accordingly .

So

```
DROP EXTERNAL TABLE [logdata]
CREATE EXTERNAL TABLE [logdata]
(
    [Correlation id] [varchar](200) NULL,
    [Operation name] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Event category] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [varchar](500) NULL,
    [Subscription] [varchar](200) NULL,
    [Event initiated by] [varchar](1000) NULL,
    [Resource type] [varchar](1000) NULL,
    [Resource group] [varchar](1000) NULL,
    [Resource] [varchar](2000) NULL
)
WITH (
    LOCATION = '/Log.csv',
    DATA_SOURCE = log_data_csv,
    FILE_FORMAT = TextFileFormat
)
```

Pausing SQL Pool/Dedicated SQL Pool: When we are not working on dedicated SQL Pool, we can pause the SQL Pool. Synapse workspace → Analytics → SQL Pools → open db → click on pause.

This will pause compute power that is being used for the pool itself. This is the majority of the cost will occupy by compute power here. When we persist data using tables, you will still be charged for the storage, but majority of the cost you can actually minimize by just pausing the pool.

LOADING DATA IN A SQL Data warehouse/DEDICATED SQL POOL IN AZURE SYNAPSE:

Normally in SQL Data base, we will use insert commands to insert a row or to insert multiple rows based on a transaction. But with a SQL data warehouse, this is used to store historical data where you want to perform analysis. So therein, you would, again, create your tables as you normally would do with the create table command. But when it comes on to writing data, you would load data in large chunks. You probably have an initial load of your data and then do delta loads. So, for example, if you had three years' worth of data, you'd first initially load that data into your SQL data warehouse, and then maybe on a daily basis whatever is the added data on your source system, you would go in and perform a delta load. Whatever are the changes, load that change onto your SQL data warehouse. So, one thing is important is loading your data and the other is how do you read your data? And again, you can load your data from your semi-structured data files or from your structured data. So, it could be from a SQL database, it could be from your JSON-based files /CSV-based files/ Parquet-based files.

LOADING DATA INTO A TABLE - COPY COMMAND - CSV:

Copy Statement in Azure Synapse is to actually copy data from an external storage accounts. This provides the most flexibility when it comes on to high-throughput data ingestion into Azure Synapse itself.

```
CREATE TABLE [pool_logdata]
(
    [Correlation id] [varchar](200) NULL,
    [Operation name] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Event category] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
    [Event initiated by] [varchar](1000) NULL,
    [Resource type] [varchar](1000) NULL,
    [Resource group] [varchar](1000) NULL,
    [Resource] [varchar](2000) NULL
)
COPY INTO [pool_logdata] FROM 'https://datalake244434.blob.core.windows.net/csv/Log.csv'
WITH(
    FILE_TYPE= 'CSV',
    CREDENTIAL =(IDENTITY ='Storage Account Key',SECRET ='
vDV2bSKSR44lbE6x05HtFz57DvIK3O2WNkb11te+H+GrBjeXCojnHjiTw3KdYBWXRJSAnOAZNdGB+AStAasz
8w=='),
    FIRSTROW=2)

SELECT * FROM [pool_logdata]
```

LOADING DATA INTO A TABLE - COPY COMMAND - PARQUET:

```
CREATE TABLE [pool_logdata_parquet]
(
    [Correlationid] [varchar](200) NULL,
    [Operationname] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Eventcategory] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [varchar](500) NULL,
    [Subscription] [varchar](200) NULL,
    [Eventinitiatedby] [varchar](1000) NULL,
    [Resourcetype] [varchar](1000) NULL,
    [Resourcegroup] [varchar](1000) NULL,
    [Resource] [varchar](2000) NULL)

COPY                                INTO                                [pool_logdata_parquet]                                FROM
'https://datalake244434.blob.core.windows.net/parquet/log.parquet'
WITH(
    FILE_TYPE='PARQUET',
```



```
CREDENTIAL=(IDENTITY='Storage Account  
Key',SECRET='vDV2bSKSR44IbE6x05HtFz57DvIK3O2WNkb11te+H+GrBjeXCojnHjiTw3KdYBWXJRSAAnOAZN  
dgB+AStAasz8w==') )
```

```
SELECT * FROM [pool_logdata_parquet] -- we will get data
```

SELECT * FROM [logdata_parquet] -- this is a external table created in dedicated SQL pool (pool DB) , if we delete Log.parquet file which is present in ADLS Parquet container, and if we do select query again on logdata_parquet- external table , we won't get data because data is read from ADLS. If we do select query on normal table pool_logdata_parquet , records will be fetched. That is the difference between external table and normal table.

BULK LOADING IN AZURE SYNAPSE:

For example If we go to Data section from LHS → Linked → click on ADLS Storage Account → CSV Container → Log.csv → right click select new SQL Script → select bulk load , new wizard it will ask below details to enter.

Field terminator : comma ,

Row terminator : \n or \r\n

First row : 2 enable infer columns

Field quote: Default(double quote “)

Date format : session default

Compression : None

Max String Length* : 4000

Select Linked Service : DLS Gen2 account

Container : container name

Click on continue

Select SQL Pool : pooldb (dedicated SQL Pool)

Select a data base : pooldb

Target table: create new/existing (can create new table or basically load it into an existing table.

If we select existing target table, next field will be Load data: Automatically or using SQL Script, if we select Using SQL Script, click on open script. we can see select query here. When doing bulk load, we will actually use pipeline feature that's available in azure synapse itself.

LOADING DATA USING PolyBase: Copy command is a good approach used for loading data into our tables. PolyBase is used to extract data from our Azure Data Lake Gen2 Storage account and then load it on to tables in our SQL Data warehouse. Again, the approach to PolyBase is same to what we've seen in terms of external tables. So, we need to create a database scope credential, an external data source and then you create an external table and then you create a normal table from the external table itself. So, this is the approach when it comes onto using PolyBase. Now with PolyBase, it's much faster when you want to load data from your external data sources, such as your Azure Data Lake Gen two storage accounts.

We already had our database scope credential, external file format, external data source and we already had our external table in place so we could easily go ahead and create our new table. if you don't have any of these artifacts in place, go ahead to create the database scope credential, to create the external data source, to create the external file format and then to create the external table itself.

```
SELECT * FROM sys.database_scoped_credentials
```

```
SELECT * FROM sys.external_data_sources
```

```
SELECT * FROM sys.external_file_formats
```

```

SELECT * FROM [logdata_parquet] -- external table
DROP TABLE [pool_logdata_parquet] -- normal table created one dropped just to show how to create
table from external table using distribution command.
CREATE TABLE [pool_logdata_parquet]
WITH
(
DISTRIBUTION = ROUND_ROBIN
)
AS
SELECT *
FROM [logdata_parquet];

```

COPY DATA FROM ADLS Gen2 STORAGE ACCOUNT ONTO AZURE SYNAPSE USING PIPELINE FEATURE:

If we want to delete data from normal table, we will use delete command.

```

DELETE FROM [pool_logdata]
Select * from [pool_logdata] --now no data returned from query

```

You can use these pipelines to actually extract data from your source, perform transformation and then load the transform data into your destination. Now what we want to do is I want to extract the data from our CSV based file and copy that data, load that data into a table in my dedicated SQL pool. So far, we have looked at various ways in which we can copy the data with copy command, PolyBase. We can now use this kind of pipeline, which is an easy way also.

Now go to Integrate Section, click on +, click on Copy Data tool, a wizard will appear with properties, source (where do I want to copy the data from?), my destination (where do I want to copy the data to) and with various settings.

Copy Data tool is used to perform a one time or scheduled data load from 90+ data sources.

Copy Data tool Wizard:

- 1) **Properties:** Here we will select copy data task type and configure task schedule here
 - **Task Type:** 2 options will be there, Built-in copy task (You will get single pipeline to copy data from 90+ data sources easily) and Metadata-driven copy task (you will get parameterized pipelines which can read metadata from an external source to load data at a large scale. We will select built-in copy task now.
 - **Task Cadence or Task Schedule:** (Run once now/schedule or tumbling window), we selected run once now.

Click on next.

- 2) **Source:**
 - **Dataset:**
 - i. **Source Type:**
 - ii. **Connection:** In source dataset we will specify connection and source type. We will create connection onto a source(ADLS Gen2 /dedicated SQL Pool db's).
 - iii. **Integration runtime:** AutoResolveIntegrationRuntime
 - iv. **File/Folder:** We will select ADLS Gen2 container file path depending on what data we want to move.
 - v. **Enable Recursively check box.**
 - **Configuration:**
 - i. **File Format Settings:** we are creating a dataset which is basically my source data. we are creating a dataset which is basically my source data. Here, it has

detected that the file format is indeed a delimited text wherein each column is separated by a comma, and each row is separated by the new line feed character and the first row is a header row. Click on the next button.

3) Destination:

- **Dataset:**
 - I. **Destination Type:**
 - II. **Connection:** we will select destination path (selected poolddb that is my dedicated SQL pool). It will automatically create a new table, but we can also select existing table that we have. Click on next.
- **Configuration:** Column mapping from source on to destination. Click on next.

4) Settings:

- **Settings :** Enter name and description for the copy data task ,more options for data movement.
 - i. **Task Name:** We will give name for the pipeline. Pipeline Name: Copy-StorageAccount.
 - ii. **Task Description:**
 - iii. **Fault Tolerance:**
 - iv. **Enable Logging**
 - v. **Enable Staging**
 - vi. **Copy Method:** different copy methods in place. Copy command/polybase/Bulk insert/upsert. Selected bulk insert here -easiest way to go ahead and insert data.
 - vii. **Bulk Insert Table Lock:** Yes/No. we selected No
 - viii. **Maximum Data Integration Unit:** Auto
 - ix. **Degree of Copy Parallelism:** blank box with edit checkbox enabled. click on next

5) Review & Finish: click on review, next and deployment will happen.

Deployment Steps Order:

- **Validating Copy runtime environment**
- **Creating Datasets**
- **Creating Pipelines**
- **Running Pipelines**

Click on finish.

Now go to integrate section → Go to pipelines → our created pipeline Copy-StorageAccount will be there

Go to Monitor Section → Integration menu → select Pipeline runs → Copy-StorageAccount pipeline run is succeeded.

Now go to develop section → open new SQL script. Select * from [pool_logdata], now records was fetched.

With the help of a pipeline, it could automatically take now the data that's stored in our CSV file in our Azure Data Lake Gen 2 storage account, and it was able to take it onto the destination table in our dedicated SQL pool.

LOADING DATA INTO PIPELINES FROM AZURE SQL DATABASE:

With the help of a pipeline, we can copy data from an existing table that you have in a SQL Database onto your SQL Datawarehouse.

Integrate → click on + → Copy Data Tool

Source: If no connection is available from drop down to select, we can click on new connection. Select Azure SQL Database .

Name: we can enter here SQL database name.

Server Name: we will select server name here.

Database name: we will enter dB name here.

Authentication type: SQL authentication

Username: we will connect on to data base using data base credentials

Password: enter dB password.

We will click on Test Connection . we can then test the connection to make sure that azure synapse can indeed connect onto the SQL database.

If connection is unsuccessful, go to SQL Database service → click on db name → click on set server firewall, In networking tab, make sure checkbox is checked (Allow Azure Services and Resources to access this server) and click on save button.

Now we need to select source tables: from existing tables or via use query.

We can select existing table and click on next.

Enter destination details,

Connection: we will select pooldb (dedicated SQL Pool db)

Same as above ADLS example, we will select settings, review, and finish.

With the use of pipeline, we can easily transfer an entire table from a source system such as Azure SQL Database onto our Azure Synapse Workspace onto our dedicated SQL Pool.

DESIGNING DATAWAREHOUSE:

When we design our tables in our data warehouse, we design them as fact tables and dimension tables. Fact table will always contain aggregated values (sum, min, max) whichever represented in number form is called as facts. Whichever data is of aggregated data or a summary of this dimension we call it as fact. Dimension is nothing but where which will give description for these facts.

Datawarehouse is nothing but database. For e.g., created table ABC in DWH, data from ADLS gen2 CSV file is loaded onto ABC table. and data from Azure SQL database can also be uploaded to same ABC table.

When we are transferring data from a source system, we have to ensure that we only copy columns of data that are required for analysis purpose. You have to ensure that you also clean your data. You don't want to have any sort of records which are probably null values that may not make sense when it comes onto analysis purpose. When it comes onto your dimension tables, you have to understand that your data for your dimension tables could come in from different data sources. So your data could come in from not only a relational database, it could also come in from let's say data stored in Azure Data Lake Gen2 storage account or from another data source altogether. In dimensional table as well you need to have a unique identifier for each row within the table.

Source System1

ProductId	ProductName	Product price
1	AZ-900	19
2	DP-900	23
3	DP-203	25

Source System2:

ProductId	ProductName	Product price
1	BookA	19
2	BookB	23
3	BookC	25

One Product Dimensional Table:

ItemKey	ProductId	ProductName	Product price
1	1	AZ-900	19
2	2	DP-900	23
3	3	DP-203	25
4	1	BookA	19
5	2	BookB	23
6	3	BookC	25

In One Product Dimensional table we can see item key 1 2 3 and 1 2 3. It does not help to uniquely identify the products within the dimension table. In such a case, we can introduce new key in our dimension table known as Surrogate key. Surrogate key is used to uniquely identify the row within the table. In Azure Synapse, actually the identity column feature is used to generate this unique ID as the surrogate ID. and in this case, The product ID is also known as the business key. the product ID, which is coming in from the source system, is known as the business key. And the item key which you are generating to uniquely identify the rows within a particular table is known as the surrogate key.

BUILDING A FACT TABLE: Using Microsoft SQL Server Management Studio, we can connect on to both Azure SQL DB and our SQL Datawarehouse.

We can create fact table from 2 different source tables by join query using view command as shown below. If we want to select only few columns from source tables and want to perform analysis only on those columns means, we can enter only those columns in select as shown below.

--- we will create view in Azure SQL Database.

```
CREATE VIEW [Sales_Fact_View]
```

```
AS
```

```
        SELECT      dt.[ProductID],dt.[SalesOrderID],dt.[OrderQty],dt.[UnitPrice],hd.[OrderDate],hd.  
[CustomerID],hd.[TaxAmt]  
FROM [SalesLT].[SalesOrderDetail] dt  
LEFT JOIN [SalesLT].[SalesOrderHeader] hd
```

```
ON dt.[SalesOrderID]=hd.[SalesOrderID]
---creating a table now within the SQL Database from the view.
```

```
SELECT [ProductID],[SalesOrderID],[CustomerID],[OrderQty],[UnitPrice],[OrderDate],[TaxAmt]
INTO [FactSales]
FROM [Sales_Fact_View]
```

```
SELECT * FROM [FactSales]
```

if you don't want to disturb your source system, you can directly use a pipeline to take the data from your source system, perform the transformation of only selecting certain rows/columns of information based on the join of multiple tables. And then transfer that information directly onto your SQL Datawarehouse.

BUILDING A DIMENSION TABLE:

```
CREATE VIEW Customer_view
AS
SELECT ct.[CustomerID],ct.[CompanyName],ct.[SalesPerson]
FROM [SalesLT].[Customer] as ct
```

-- Lets create a customer dimension table

```
SELECT [CustomerID],[CompanyName],[SalesPerson]
INTO DimCustomer
FROM Customer_view
```

-- Lets build a view for the products

```
CREATE VIEW Product_view
AS
SELECT prod.[ProductID],prod.[Name] as ProductName,model.[ProductModelID],model.[Name] as
ProductModelName,category.[ProductcategoryID],category.[Name] AS ProductCategoryName
FROM [SalesLT].[Product] prod
LEFT JOIN [SalesLT].[ProductModel] model ON prod.[ProductModelID] = model.[ProductModelID]
LEFT JOIN [SalesLT].[ProductCategory] category ON prod.[ProductcategoryID]=category.
[ProductcategoryID]
```

-- Lets create a product dimension table

```
SELECT [ProductID],[ProductModelID],[ProductcategoryID],[ProductName],[ProductModelName],
[ProductCategoryName]
INTO DimProduct
FROM Product_view
```

-- If you want to drop the views and the tables

DROP VIEW Customer_view

DROP TABLE DimCustomer

DROP VIEW Product_view

DROP TABLE DimProduct

TRANSFER DATA TO OUR SQL POOL: We will transfer data onto dedicated sql pool simply via pipeline by selecting source (Azure SQL Database) and destination details (dedicated sql pool) and create pipeline. We can select multiple source tables (2 dim and 1 fact table) at a time while creating pipeline.

In traditional sql based engine, there is a foreign key constraint that establishes relationship between 2 tables. when it comes onto Azure Synapse, the dedicated SQL pool, and the SQL data warehouse, you don't have the concept of defining foreign key. We will map data across 2 tables using join clause only instead of foreign key in dedicated sql pool.

Even we didn't mentioned foreign keys for tables in Azure dedicated sql pool within synapse, power bi automatically understood there is a relationship between fact and dimension tables. when you go onto the pipeline, which transfer the data, it would've not transfer the foreign key constraint, if there was any in the source system because that's not basically supported in Azure Synapse.

AZURE SYNAPSE ARCHITECTURE:

When you write the query and target the query against table in the dedicated SQL pool, the query actually goes onto control node. So, this is a compute node that has control over the other compute nodes that actually do the work on the query. It is a controlled node's responsibility to distribute the query that's fired by the user against the compute nodes that are part of the dedicated SQL pool. The actual data in the table resides on Azure Storage. When it comes onto a traditional SQL database, the workings of the database engine depend, obviously, on the compute power for processing the queries, and the data itself needs to reside somewhere. It needs to reside on some sort of disk. **In the case of Azure Synapse, you have various compute nodes that are responsible for handling the queries. The query itself is submitted to the SQL endpoint. The SQL endpoint forwards onto the control node first, and then the control node, distributes the query against the compute nodes. The data itself resides in Azure Storage. So, this is like your disk holding the data within the tables.**

what is the purpose of having multiple compute nodes?

When you are looking at working with a SQL data warehouse, you are looking at performing analysis on terabytes of data on millions and millions of rows. in the actual-world scenario, when you want to use a SQL data warehouse, a company will be having millions and millions of rows in the SQL data warehouse, even billions of rows within the SQL data warehouse itself. Now, if you fire a query to perform analysis, a SQL query to perform analysis, on those billions of rows, obviously, if you have multiple compute nodes actually trying to work on the data, trying to, let's say, fetch the data for a select statement in the query, if you have these multiple compute nodes working in parallel, then you'll get the result faster because you are trying to split now the workload of the data across multiple compute nodes. So, when it comes onto Azure Synapse, the compute and the storage components are separate. Here, both the compute and storage can scale independently. We've seen that with the dedicated SQL pool, you can also pause the compute layer. You'll still pay for the underlying storage. Now, when it comes onto the dedicated SQL pool, the compute power is charged in terms of something known as DWU. That's data warehousing units. Data warehousing units is a combination of compute, memory and IOPS. Now, when it comes onto storage, your data is actually split into multiple distributions. This is actually used to optimize on

performance. By default, there are 60 distributions in a dedicated SQL pool. If you want to look at the distributions for a table, you can go ahead and basically use the below statement against the table itself.

DBCC PDW_SHOWSPACEUSED('dbo].[FactSales]') if we run this query, there is DISTRIBUTION_ID column. For this column if we see there are indeed 60 distributions. Every table is gonna be split it in to 60 distributions.

SUMMARY OF AZURE SYNAPSE ARCHITECTURE:

that we have a control node that is used to accept the SQL queries that is fired against SQL data warehouse. We have the compute nodes that are used to actually process the data. The control node actually sends the query onto the compute nodes to process the query accordingly. The data in your table is stored in Azure Storage, and your table data, by default, is split into 60 distributions.

TABLE TYPES:

Hash-distributed tables → beneficial to use on large tables .

Round-Robin distributed tables

Replicated tables

HASH-DISTRIBUTION:

By default, when you create a table in Azure Synapse if you don't specify any sort of distribution, the distribution type that is used is a round-robin distribution. Let's take a very simple example wherein we have a table that has the information about the Order ID, the Course ID, and the Quantity. In this table, I only have, just for the sake of simplicity, three rows of data.

When you create a table based on hash distribution you tell the SQL pool during the creation table time how do you want to hash the rows in the table? You will tell which column information, which column value, is going be used for distributing the data. So, let's say that you tell the SQL pool that when you create the table, please distribute my data based on the Order ID. So internally every table has 60 distributions. Here again, for the sake of simplicity I'm only taking three distributions into context. So, your rows, the first row of data which has a value of, let's say, 01, will be sent onto the first distribution. The second row of 02 will go onto the second, so on and so forth. internally, the SQL pool will use a hashing function to decide which row should go onto which distribution.

what is the benefit of having these hash distributions being hashed based on the data?

So, let's say in this simple case you write a SQL query wherein you want to only select the row where the column ID is equal to 01. Then automatically it'll use a hashing function and the SQL pool will just go onto the distribution, continue the row for 01. So instead of searching all of the distributions, all of the table, remember you might be having a billion rows on the table, just to search for a particular set, a data set within that large table, and scanning the entire table will just take too much time. **What Azure Synapse can do is it can actually drill down onto a particular partition and fetch the data accordingly.**

ROUND-ROBIN DISTRIBUTION: Now in round-robin, the rows of data are just added one by one into each distribution. You are not telling the SQL pool, please use any hashing function to distribute the data. it's just one after the other. Table has 3 rows, 1 row to 1 distribution, 2nd row to 2nd distribution, 3rd row to 3rd distribution. If you had a fourth row of data, it would go into the first distribution. If you had 5th row of data, it'll go into the 2nd distribution. So, a round-robin fashion.

DIFFERENCE BETWEEN ROUND-ROBIN DISTRIBUTED TABLE AND HASH DISTRIBUTED TABLE:

let's say that you had column data that was used for hashing based on the Course ID, then let's say, in this particular table, the last two rows of data have the same Course ID. So, both of these rows are coming onto the same distribution. The third distribution is currently empty in hash distributed but if it was a case of round-robin, all three distributions would have data. So, this is the difference between the round-robin distributed tables and the hash distributed tables.

Now, round-robin distributed tables actually provide fast performance when it comes onto lowering data in staging tables. And here the data is distributed across the table.

REPLICATED TABLES: This provides the fastest performance when it comes onto small tables. Here each compute node actually caches a full copy of the table. Now, even though I've kind of mapped each table for each distribution, so it's like each distribution having a copy of the data.

each compute node in Azure Synapse in your dedicated compute, in your dedicated SQL pool, will have access to the entire table. So instead of actually searching a particular distribution the compute node will only have access to the table.

So, you can use hash distribution when it comes onto your fact tables, and you can use replicate tables when it comes onto your dimension tables. Since your dimension tables are smaller when compared to your fact tables, you can have them replicate across the various compute nodes. If you try to replicate the fact tables, it'll just be too much. I mean, you're trying to replicate billions of rows across each compute node. Doesn't make sense. But since the dimension tables are smaller you can make use of them for replicated tables.

STAGING TABLE: we could transfer data using pipeline in Azure Synapse from, let's say, a storage account or Azure SQL database onto our SQL data warehouse. So there, what we are doing is we are extracting data from a source system and then we are loading the data in a data warehouse, and we can perform analysis on the data (via visualization tool pbi tool). Now, in between, we can also transform our data, which is very important. So, from the source system, you don't want to have all of the columns of data. You wanna ensure that maybe duplicate values are not present in the source data. You wanna ensure that null values are taken care of in the source data. Different things that you need to do when it comes onto the source data itself. So at this point in time, you wanna transform your data. So, you extract your data, you transform your data, and you load your data. This entire process is known as the ETL process: the extract, transform, and load process. So where does the staging area come in? See, when you have your source system, let's say you have an Azure SQL database. You have an application which has been used by users that is working with the tables in the Azure SQL database. Now in our case, what we had done is we had created something known as a view when it came onto building our fact tables and dimension tables. We built a view in the source system itself in the SQL database. Then we built another table there and then we transfer the table and then we transfer the tables onto the SQL data warehouse. this is probably not the best approach is because you are putting an impact, you are making a change onto the source system. And in your production-based environment, you don't want to make such changes. You don't want to impact the production database itself. So normally, what you do is you would take your data, maybe your tables, the entire table, onto a separate area, let's say a staging area. And then from there, you'll make whatever transformation needs are required and then transfer them onto a SQL data warehouse. In Azure Synapse as well, sometimes you might have a faster loading time when you first load the data into staging tables and then copy the data from the staging table onto the data warehouse itself.

COMPUTE TO STORAGE/DATA DISTRIBUTION:

when we fire queries against let's say, a dedicated SQL pool the queries are first received by the control node and then the control node passes the query onto the various compute node. The various compute nodes are gonna work with your data distributions, which are stored in Azure storage for each of your tables. And each table is divided into 60 distributions. So now in this case, let's say that you have data that's split across 60 distributions. This is your data for one of your tables. Let's say that you have a hash distributed table. So you have data in all of these logical distributions. So, note that internally the dedicated SQL pool will understand how to put a row of data into its respective distribution and, it gives information on how to retrieve the rows of information from each distribution. It knows, based on hashing function for a hash distributed table. Now, what about the compute nodes? So, the control node will send the response onto your compute nodes. In your dedicated SQL pool, you can also have 60 compute nodes in place. You can start with one and you can go all up to 60.

If you have 60 compute nodes, you'll have a one-to-one mapping between each compute node and each distribution. This is where it becomes effective to fire queries and basically analyze your data across the distributions. So, in this case, the control node can send the query onto each compute node. Each compute node can target, can attack each distribution separately. Because data within a table is split across multiple distributions. So, one compute node going from one distribution onto another, fetching the data and then giving it back to the control node, and then the control node gives the response back to the user. all of these compute nodes can work in parallel can attack each distribution, fetch the required data and then everything is combined, sent across to the control node and sent across to the user. So, it's kind of a divide and conquer approach. You wanna ensure that you get the fastest performance by putting your work being done in parallel over here. So that's the benefit of having these distributions in place. Because you have your data that's split across. You can work on them independently and then they can be merged together in terms of the results set and then send back onto the user. Now in our case, we know that our table has 60 distributions. But does dedicated SQL pool have 60 compute nodes? No. We just have one compute node. The number of compute nodes that get assigned to you depends upon the data warehousing unit that you choose for your pool. Now we chose the most lowest setting when it comes onto the data warehousing unit (DWU100 → 1 compute node). So, we do get 60 distributions, but we only get one compute node.

It's only when you go onto the maximum (DWU30000c has 60 compute nodes). So, you can see that number of compute nodes increases based on the data warehousing unit that you choose. So obviously when you go onto the maximum, you get 60 compute nodes, and because you'll have 60 distributions by default, the number of compute nodes per distribution is going to be one. So, when you have a very large data warehouse and you wanna perform analysis of your data it's very important to know finding the right size when it comes onto data warehousing units.

SERVERLESS SQL POOL VS DEDICATED SQL POOL:

with the serverless SQL pool, you only can create external tables. Your data is in Azure Data Lake, let's say Azure Data Lake Gen 2 storage account. It's only the structure of your table that is in the serverless SQL pool. And you will do this for the initial analysis of your data in the data lake in serverless sql pool. If you're performing analysis, if business users are performing analysis in serverless sql pool, this can become quite slow because you still have your data somewhere else and the query, the results set has to be fetched from the data in an external data source. All of this will take time. Whereas when it comes onto a dedicated SQL pool, all of the data is already present in Azure storage that is connected with the compute node in the dedicated SQL pool. Everything is close together. We can't create distributed

tables No hash distributed tables, no replicated tables, no round robin tables in serverless sql pool whereas we can create distributed tables in dedicated sql pool.

ROUND-ROBIN TABLES: when you create a table without specifying the table type, it'll create the table as a round robin distribute table by default. But, if you wanna confirm on the distribution type of the tables, you can execute below query.

```
SELECT OBJECT_NAME( object_id ) AS table_name,  
       * FROM sys.pdw_table_distribution_properties; we will get table distribution type.
```

-- If you execute the below query

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

ORDER BY [CustomerID] when we run this query, if I want to know data operation type how this query running in back end, we will go to Integrate → Activities (SQL Requests) → Request Content field (whatever queries/requests we are firing on db those information is showing here) . if we click on Request ID field, we will see operation types there.

CREATING A HASH-DISTRIBUTED TABLES:

-- Let's drop the existing table

```
DROP TABLE [dbo].[FactSales]
```

-- Now we want to create a hash-distributed table and set the hash-based column as the Customer ID

```
CREATE TABLE [dbo].[FactSales](  
    [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  
(  
    DISTRIBUTION = HASH (CustomerID)  
)
```

-- To see the distribution on the table

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

-- If you execute the below query

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

ORDER BY [CustomerID] -- if we execute this query , result will be fetched with in a fractions of seconds which can be tracked in monitor section → Activities → SQL Requests → in operation type → only return operation is shown.

CREATING REPLICATED TABLES:

```
CREATE TABLE [dbo].[FactSales](
    [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
(
    DISTRIBUTION = REPLICATE
)
```

TABLE DESIGNING: your fact tables tend to be large and normally you will have them as hash distributed tables. You will decide on the column that will be used for hashing purposes.

Your dimension tables tend to be smaller hence it's good to use replicated tables and normally use round-robin tables for your staging tables because it's easy to load. See, in round-robin Azure Synapse doesn't need to think about how to allocate the rows. One after the other it will start allocating it onto a distribution in a round-robin fashion. So, it becomes easier to load the data. If you are to load the data initially in a fact table which is based on hash distribution it could take time, right? Because Azure Synapse first has decided on the distribution based on the hashing of the column value. That's why normally it's easier to first load your data into a staging table and then move that data onto your hash distributed fact table.

DATA MOVEMENT SERVICE: Let's take as an example that you have two tables in place. One is an orders table (a fact table) and another is a dimension table (Instructors table). So, let's say you have a course platform selling courses. You have tables about your orders, instructors, your courses, etc. This is in your SQL data warehouse. Now, if you base your fact tables on hash distribution based on a particular course category ID, so let's say that each course category ID is assigned onto a specific distribution. Let's assume just for now, we know that there are 60 distributions. We know that you can have 60 compute nodes (for DW30000c). Let's imagine that we just have three distributions, and we have three compute nodes that are managing each distribution. Now, let's say that you create the dimension table for the instructor based on round-robin distribution. So, the rows of the table are basically disappeared across the distributions itself. Now you want to perform a join on the fact and dimension table to let's say to get the popular instructors based on a particular course category. So, from the perspective of the fact table when you're joining the fact table with a particular course category. Let's say it's for the first course category, so that CPU (compute node) can target the data just in its distribution. You are trying to filter based on a particular course category. But then for a particular course category, you could be having multiple instructors and your instructors are distributed across its own distributions, right? For this table. So now what happens is that since your primary compute node is working with a particular distribution, when it comes onto the fact table for your orders in order to join it onto the instructor table, those instructors which fall in that course category might be across multiple distributions. So now

in such a case, the data movement service which is available in Azure Synapse has to move the data relevant

to those instructors onto that compute node. That compute node will anyway take the data in this distribution but needs to also take the data from the other distributions for the dimension table. So please note here, when I have actually put these symbols, this is not the entire table, just for the sake of simplicity, just take this as a small dataset. So, let's say this is one dataset of your table based on a particular course category. Another one and another one. In terms of the instructor table, you have one data set in a round-robin fashion that has been done for the entire instructor table. So, I told you that if you are targeting a join query against a particular course category, at least the CPU can target only that distribution when it comes onto the fact table. But since for your instructor table, there is no defined way in which the rows have been distributed,

it's in a round-robin fashion. So, the data movement service has to go onto each distribution on that site for that table, fetch the information and move it onto the compute node so that the compute node can perform the joint of that data and the data in the fact table. So, this data movement can be an expensive operation.

Now, it is impossible to completely remove the fact that you have data movement in place, but you can minimize it if you know the queries that you want to execute and then build your tables accordingly. Whereas if you were to create, your instructor dimension table has a replicated table. So here, this is not a portion of the instructor table. Think of this as the entire instructor table that will be available for each compute node. So, because of this now, when you perform that same join of the tables for a particular course category, since each compute node has a replicated copy of the instructor table, there is no data movement service, it can take all of the data

from that replicate instructor table and just work with the data in its distribution itself.

CREATING HASH DISTRIBUTION FOR MULTIPLE COLUMNS: We can create hash distribution on multiple columns as shown below. if you want to go ahead and create the facts sales table based on hash distribution, based on the customer ID, product ID, you can do that and internally Azure synapse will distribute the data accordingly. if we wanna go ahead and create below table, we also have to execute this command,

```
ALTER DATABASE SCOPED CONFIGURATION SET DW_COMPATIBILITY_LEVEL = 50
```

which is an altered database command. We have to set the compatibility level and after you can we go ahead and create the table based on the hash distribution of multiple columns.

```
DROP TABLE [dbo].[FactSales] -- table already there so dropped.
```

```
CREATE TABLE [dbo].[FactSales](
    [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
(
```

```
DISTRIBUTION = HASH ([CustomerID],[ProductID])
)
```

GOOD PRACTICES WHEN DESIGNING TABLES: When to use what distributions?

Hash-Distribution: This improves query performance on large tables. Choose right column that data gets distributed across the distributions.

Round-Robin Distribution: This helps in improving loading speed.

Data Movement: With hash-distribution, SQL Analytics has knowledge on the rows in the distribution. Choose a column that minimizes data movement – is used in JOIN, GROUP BY clauses.

Choose Round-Robin: No joining key, no good candidate for hash distribution.

Fact Tables: Best to use hash distribution

Dimension: Best to use replicated tables.

SURROGATE KEYS FOR DIMENSION TABLES: When you have data coming in from different sources into the same table, in order to uniquely identify the rows within a table, you might create an additional column. This could be a simple column with numbers that can be helped to that can help to uniquely identify the rows within the table. So this extra column of data is basically the surrogate key for the table itself. So, in this case, for example if you have a table wherein the product ID is coming in from the main source table, this is normally referred to as the business key or the alternate key. This is a primary key in the source system. But if you wanna have an additional key, you can go ahead and do this as a surrogate key. In Azure Synapse, you can actually make use of the identity column feature when it comes onto your dedicated SQL pool tables.

```
DROP TABLE [dbo].[DimProduct]
```

```
CREATE TABLE [dbo].[DimProduct](
    [ProductSK] [int] IDENTITY(1,1) NOT NULL,
    [ProductID] [int] NOT NULL,
    [ProductModelID] [int] NOT NULL,
    [ProductcategoryID] [int] NOT NULL,
    [ProductName] varchar(50) NOT NULL,
    [ProductModelName] varchar(50) NULL,
    [ProductCategoryName] varchar(50) NULL
)
```

extra column [product SK] is surrogate key. Identity(1,1) is to identify identity column and please generate a unique ID for me every time. A unique number for every row. So we don't need to add any values for this particular column. When rows are being inserted into the table, this will automatically be added by the dedicate SQL pool.

Now after table is created, go integrate → + → copy data tool → enter source details,

in destination details of column mapping wizard, for source column, mapping source is empty for ProductSK column(surrogate key) which is there in destination table. So we simply delete surrogate key column from column mapping because surrogate key column(ProductSK) , values are gonna be automatically generated by the dedicated sql pool.

SELECT * FROM [dbo].[DimProduct] -- if we query this productSK column data is not in order, so we will query with orderby clause.

SELECT * FROM [dbo].[DimProduct]

ORDER BY [ProductSK] → if I query this, for an identity column, it should ideally start from 1 and then increment accordingly. But in Azure Synapse, when they implement this identity feature the value is actually based on the distribution. So, they have a different way in which they assign a value for the product SK column.

SLOWLY CHANGING DIMENSIONS:

when it comes on to your fact tables normally what will happen is that data will just get appended or added onto your fact table. Even when it comes on to your dimension tables as well, if you go onto, let's say your product dimension table, yes, when new products are added onto the source system, they will be added onto the dimension product table. But normally, when it comes on to updates for existing columns, existing rows of data, it probably won't happen onto your fact sales table because that will just skew the facts that you have. But when it comes on to, let's say, your dimension tables, you might have slight changes in the source data. For example, if you look at our dimension product table. Let's say that there is a slight change in the product name that's linked onto the product ID in the source system in let's say, our SQL database. We have to ensure that the update is propagated onto our dimension table. The way that we actually handle this change is known as slowly changing dimension tables. We have different types, different ways we can handle this.

Type-1: just to update the row as it is, update the changes as it is. If it does not make a big impact on how we view the data in relation to our fact table, then we can keep it as it is.

Type-2: Here we will keep both older value and newer value in the dimension table. Maybe the business wants to see the data in the fact table that relates on to the product with the old name, and maybe the data for the product with the new name. In this case, it is known as type two when it comes on to slowly changing dimensions.

ProductSK	ProductID	ProductName	StartDate	EndDate	isCurrent
1	1	ProductA	2021-03-20	2021-04-20	False
2	1	ProductAA	2021-04-21	9999-12-31	True

you can have two additional columns that specify what is a start and the end date for each column. So, between the start and the end date, this was the current product name or the old product name. What is the new product name is from the start end date to let's say, a beyond end date (9999-12-31). And you can have another column(isCurrent) that tells whether the current product is the true one or false one that is being used in terms of product name.

Type-3: Here instead of having multiple rows to signify changes, we now have additional column to signify the changes. So instead of having, I said additional rows, you can have one column of the original name, another one with a changed name, and another one saying what is the effective date of the change.

ProductSK	ProductID	Original Name	Changed Name	EffectiveDate
-----------	-----------	---------------	--------------	---------------

1	1	ProductA	ProductAA	2021-04-20
---	---	----------	-----------	------------

INDEXES IN AZURE SYNAPSE: used to improve query performance.

When it comes on to traditional normal data bases, when the database server stores the data within the table, it's stored on the disk. When the database server needs to work with the data, the CPU on the database server needs to load the data from the disk onto its memory. And then whatever needs to be done in the memory based on the query itself, and then processing is done on the data within the memory, and then it is returned as a result set onto the user. So, there's a lot that actually goes on when it comes onto searching for data.

Now, when it comes onto tables in Azure Synapse in your dedicated SQL pool, we looked at distributions. And we know that distributions are also used to effectively work with data. Distributions are helpful when you are using the join clause to join data between tables and in the group by clause. When you want to filter on queries using the WHERE clause. So normally the WHERE clause will try to filter on certain columns of data, at that point in time to make that search much more efficient, we can make use of indexes. So, we have different aspects in place to improve the overall efficiency of queries. On one hand, we have distributions, on the other hand, we have indexes.

Now by default, when it comes onto a table in the dedicated SQL pool, a clustered column store index is created. This good for large fact tables has, it provides not only better query performance, but also better compression of data. So, if you wanna have better query performance for let's say a fact table in your dedicated SQL pool, keep it as a clustered column store index. Now what is the way that the clustered column store index actually works?

If you look at a relational database, here in the relational database when the rows of data are stored in memory, they are stored in a row vise format as shown below row1, row2 in below table and row 3 , row4 at Page 2 in Memory.

Page 1 in Memory

ColumnA	ColumnB	ColumnC
Value 1	Value 2	Value 3
Value 4	Value 4	Value 5

Page 2 in Memory:

ColumnA	ColumnB	ColumnC
Value 7	Value 8	Value 9
Value 10	Value 11	Value 12

And the indexes help to fetch the rows within the table. Whereas in a data warehouse, the data is actually stored in terms of memory in a columnized format as shown below.

Page1 in memory

Page2 in memory

Page3 in memory

Column A	Column B	Column C
Value 1	Value 2	Value 3
Value 4	Value 5	Value 6
Value 7	Value 8	Value 9
Value 10	Value 11	Value 12

Because normally in your data warehouse, you'll be working with certain columns of data, in some rows of data, and you would normally want to perform aggregation towards the values in those columns. So it becomes much more effective for the underlying engine to store the data in your table in a columnized format in memory. So now we digged deep into how the data is actually stored by the engine itself.

certain cases where we can't create clustered column store indexes:

- when table contains columns of the type varchar(max), nvarchar(max), varbinary(max) etc.,
- if you wanna create temporary tables.
- If you have a table with less than 60 million rows

When to consider heap tables:

- If you wanna create temporary tables, then it's actually better to go at and consider the use of heap tables
- If you have a table with less than 60 million rows, we can consider heap tables.

In dedicated SQL pool, clustered columnstore index is automatically created, this is good for large fact tables. it provides good compression of data and better query performance.

CLUSTERED INDEXES: If we don't want to have normal clustered column store indexes, you can define normal clustered indexes. clustered indexes also improve the performance of queries that have very high selective filters that narrow down on a row of data

NON-CLUSTERED INDEXES: if you want to improve the performance of filtering of other columns of data in that same table which has a clustered index, you can then create a non-clustered index on the same table. But the non-clustered index needs additional space and processing time for the index itself.

CREATING INDEXES:

```
CREATE TABLE [pool_logdata]
(
    [Correlation id] [varchar](200) NULL,
    [Operation name] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Event category] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
    [Event initiated by] [varchar](1000) NULL,
    [Resource type] [varchar](1000) NULL,
    [Resource group] [varchar](1000) NULL,
    [Resource] [varchar](2000) NULL
)
WITH
(
    DISTRIBUTION = HASH ([Operation name]),
) -- this query will be created with clustered column store index .
```

```
DROP TABLE [pool_logdata]
CREATE TABLE [pool_logdata]
```

```
(
  [Correlation id] [varchar](200) NULL,
  [Operation name] [varchar](200) NULL,
  [Status] [varchar](100) NULL,
  [Event category] [varchar](100) NULL,
  [Level] [varchar](100) NULL,
  [Time] [datetime] NULL,
  [Subscription] [varchar](200) NULL,
  [Event initiated by] [varchar](1000) NULL,
  [Resource type] [varchar](1000) NULL,
  [Resource group] [varchar](1000) NULL,
  [Resource] [varchar](2000) NULL
)
WITH
(
  DISTRIBUTION = HASH ([Operation name]),
  CLUSTERED INDEX ([Resource type])
) → if we want to create normal clustered index for your table, we will choose clustered index in
create command.
```

If we have queries which we use where clause on resource type column, we will create clustered index on that column as shown above.

If we have other queries that are based on event category in the where clause, we can create another index non clustered index as shown below.

```
CREATE INDEX EventCategoryIndex ON [pool_logdata] ([Event category]) -- non clustered index creation.
```

WHICH LOADING METHOD TO CHOOSE ON COPYING DATA ONTO OUR TABLES IN DEDICATED SQL POOL:

When we want to copy data from source to destination (in dedicated SQL pool) via pipeline using copy data tool , we have seen copy command/ Polybase /bulk insert .

BULK INSERT: When we have less data to transfer then probably, we can go ahead and use bulk insert command. Bulk Insert method is slower than copy command/ Polybase. Polybase gives us better performance when we are loading large amounts of data.

When we are using bulk insert command, the command actually goes through the control node. The control node then kind of has the job of ensuring that the data is actually inserted with the help of the compute nodes onto the table.

POLYBASE: But with the help of PolyBase, what happens is our data movement operations go through the compute nodes directly in parallel, it does not go through the control node. So, this provides better loading times. And if you have more compute nodes, the data loading process becomes faster. Also, with PolyBase, you can create external tables and then you can import data into dedicated SQL pool, creating tables using CREATE TABLE AS SELECT Command.

COPY COMMAND: copy command also has a high throughput and goes through the compute node in the dedicated SQL pool, the copy command is simple to execute. You don't need to define the external data source, the external file format, and the external tables.

CREATING HEAP TABLES: When you wanna have a temporary landing zone for your data in the dedicated SQL pool, you could actually make use of heap tables to load the data faster. Also, if you have more than 60 million rows in your table, then it's good to consider the use of a clustered column store table. But if you have small lookup tables with less than 60 million rows, then you should consider the use of heap tables or clustered index for faster query performance.

```
CREATE TABLE [logdata]
(
    [Correlation id] [varchar](200) NULL,
    [Operation name] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Event category] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
    [Event initiated by] [varchar](1000) NULL,
    [Resource type] [varchar](1000) NULL,
    [Resource group] [varchar](1000) NULL,
    [Resource] [varchar](2000) NULL
)
WITH
(
    HEAP,
    DISTRIBUTION = ROUND_ROBIN
)
```

```
CREATE INDEX ResourceTypeIndex ON [logdata] ([Resource type])
```

TABLE PARTITIONS IN AZURE SYNAPSE: Table partitions is another way to effectively improve performance of queries that has where clause within them.

Hash distributed table is effective when you perform joins on our tables. you can actually split your data into different partitions. It's like different logical partitions. This helps to divide your data into smaller groups of data. This is different from distributions. Partitions are normally created on a date column. So, when you have a column that has date information, you'll normally create a partition on the date column. When you have queries that are based on the date column, it's effective then to make use of these partitions. Now, partitions are supported for all distribution types when it comes onto your dedicated SQL pool. So, you could have partitions for your hash distributed tables, for your round robin tables, or even for your replicated tables. It is also supported for your heap tables, your clustered column store tables, and your clustered indexes. Now, when it comes onto the clustered column store table, for optimal compression and performance, you should idly have a minimum of one million rows

per distribution and partition. So, let's say that you have a table that has 12 monthly partitions. So normally, partitions are going to be based on the date column and you have 12 monthly partitions in place. Our table already has 60 distributions. So, if you are looking at a minimum of one million rows per distribution per partition, when it comes onto the distributions itself, you should have a minimum of 60 million rows. And now when you consider that you also have 12 partitions in place, you have to multiply that figure by 12(60 million rows * 12 = 720 million rows). That means that you have an effective performance of using partitions if you have more than 720 million rows. Also, please note that you shouldn't be having too many partitions in your table. Have an effective number of partitions. Also, with the partitions itself, it becomes easier to insert and delete data because inserting and deleting data in a large data warehouse can be an expensive operation.

```
SELECT FORMAT(Time,'yyyy-MM-dd') AS dt,COUNT(*) AS 'Number of Operations' FROM [logdata]
GROUP BY FORMAT(Time,'yyyy-MM-dd') → time column has data of 2023-04-17 15:30:14.030 , so
formatting time column to string.
```

o/p for this select query after formatting for time column will be 2023-04-17

-- Let's drop the existing table if it exists

```
DROP TABLE logdata
```

-- Let's create a new table with partitions

```
CREATE TABLE [logdata]
(
    [Correlation id] [varchar](200) NULL,
    [Operation name] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Event category] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
    [Event initiated by] [varchar](1000) NULL,
    [Resource type] [varchar](1000) NULL,
    [Resource group] [varchar](1000) NULL,
    [Resource] [varchar](2000) NULL
)
WITH
(
    DISTRIBUTION = HASH ([Operation name]),
    PARTITION ( [Time] RANGE RIGHT FOR VALUES
        ('2023-01-01','2023-02-01','2023-03-01','2023-04-01'))
)
```

-- You will see 5 partitions,

Partition1: one is for those values prior to 2023-01-01 . <'2023-01-01'

Partition2: >='2023-01-01' and < '2023-02-01'

Partition3: >='2023-02-01' and < '2023-03-01'

Partition4: >='2023-03-01' and < '2023-04-01'

Partition5: >='2023-04-01'

Note: not to create too many partitions. Data will be distributed across distributions and partitions.

Now using pipeline, monitor → integration → Pipeline runs, we will rerun pipeline which we already created to copy data into our table.

when you execute a SQL query using where clause based on the date column, then the SQL engine in the dedicated SQL pool can actually narrow down to the partition that contains data within that date range.

SWITCHING PARTITIONS: Instead of deleting rows using delete statement(which is a cost effective operation) , we will make use of partitions and we can remove that data.

```
CREATE TABLE [logdata_new]
WITH
(
  DISTRIBUTION = HASH ([Operation name]),
  PARTITION ( [Time] RANGE RIGHT FOR VALUES
    ('2023-02-01','2023-03-01','2023-04-01')

  ))
AS
SELECT *
FROM logdata
WHERE 1=2 -- only jan month data will be fetched due to this condition.
```

ALTER TABLE [logdata] SWITCH PARTITION 2 TO [logdata_new] PARTITION 1; --- we can remove partition using alter command.

```
SELECT count(*) FROM [logdata_new]
SELECT FORMAT(Time,'yyyy-MM-dd') AS dt,COUNT(*) FROM logdata_new
GROUP BY FORMAT(Time,'yyyy-MM-dd')
```

-- See the data in the original table

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

READING JSON DATA FROM ADLS:

After loading json file in ADLS, we will go to Data → Linked -> ADLS Storage account → JSON container → Log.json file → right click select 100 rows , below code will be displayed.

```
SELECT TOP 100
  jsonContent
FROM
  OPENROWSET (
    BULK 'https://datalake2000233.dfs.core.windows.net/json/Log.json',
    FORMAT = 'CSV',
    FIELDQUOTE = '0x0b',
    FIELDTERMINATOR = '0x0b',
    ROWTERMINATOR = '0x0a'
  )
```

```
WITH (
    jsonContent varchar(MAX)
) AS [rows]
```

Everything is returned here as 1 column has json content , so we will keep json_value statement to extract each column name separately from json content as shown below.

```
SELECT
    JSON_VALUE(jsonContent,'$.Correlationid') As Correlationid,
    JSON_VALUE(jsonContent,'$.Operationname') AS Operationname,
    JSON_VALUE(jsonContent,'$.Status') AS Status,
    JSON_VALUE(jsonContent,'$.Eventcategory') AS Eventcategory,
    JSON_VALUE(jsonContent,'$.Level') AS Level,
    CAST(JSON_VALUE(jsonContent,'$.Time') AS datetimeoffset) AS Time,
    JSON_VALUE(jsonContent,'$.Subscription') AS Subscription,
    JSON_VALUE(jsonContent,'$.Eventinitiatedby') AS Eventinitiatedby,
    JSON_VALUE(jsonContent,'$.Resourcetype') AS Resourcetype,
    JSON_VALUE(jsonContent,'$.Resourcegroup') AS Resourcegroup,
    JSON_VALUE(jsonContent,'$.Resource') AS Resource
FROM
    OPENROWSET(
        BULK 'https://datalake2000233.dfs.core.windows.net/json/Log.json',
        FORMAT = 'CSV',
        FIELDQUOTE = '0x0b',
        FIELDTERMINATOR ='0x0b',
        ROWTERMINATOR = '0x0a'
    )
WITH (
    jsonContent varchar(MAX)
) AS [rows]
```

WINDOWING FUNCTIONS: can be used to apply a mathematical equation on a set of data that is defined within a particular window. Here you can split rows into different sets and apply aggregation for the data in each set. We will write window function by using OVER clause.

```
CREATE TABLE [logdata]
(
    [Correlation id] [varchar](200) NULL,
    [Operation name] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Event category] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
```

```

        [Event initiated by] [varchar](1000) NULL,
        [Resource type] [varchar](1000) NULL,
        [Resource group] [varchar](1000) NULL,
        [Resource] [varchar](2000) NULL
    )
WITH
(
    DISTRIBUTION = HASH ([Operation name])
)

SELECT
ROW_NUMBER() OVER(PARTITION BY [Operation name] ORDER BY [Event category]) AS "Event
Category",
[Operation name],[Event category],[Resource type],[Resource]
FROM [dbo].[logdata]
ORDER BY [Operation name]

```

Here from above query, data will be fetched by partitioning operation name, i.e., if operation name has column values as a d, c, b,, first data will be fetched for operation name a all rows, next for b all rows, next c all rows, next d all rows etc.

CASE STATEMENT USAGE:

```

SELECT distinct[Event category] FROM [dbo].[logdata]

SELECT
EventTier = CASE [Event category]
WHEN 'Administrative' THEN 'Tier 1'
WHEN 'Resource Health' THEN 'Tier 2'
WHEN 'Service Health' THEN 'Tier 3'
WHEN 'Policy' THEN 'Tier 4'
WHEN 'Recommendation' THEN 'Tier 5'
END,
COUNT([Operation name]) AS 'Count of operations'
FROM [logdata]
GROUP BY [Event category]

```

