Dedicated SQL Pool (Azure SQL Datawarehouse)

# Dedicated SQLpool is Paas service offering from Microsoft Azure, fully managed relational data warehouse as a service.

# Features:

# Highly scalable cloud based distributed relational database management system

# MPP Architecture

# Distributed storage and compute

# In-memory column store for 100x speed improvement

# Polybase

# Understanding MPP Architecture

# SMP (Symmetric multiprocessing):

|  |  |
| --- | --- |
| https://cloudblogs.microsoft.com/uploads/prod/2018/03/dpi-jul30-1.jpg | Symmetric multiprocessing system is like classical SQL Server system, usually having one large server machine.Data is stored locally.  * It is multi-processor system where processor share OS and Memory and I/O devices and connected using a common bus  The distribution of workload between processors is performed by the operating system and multiple CPU serve separate processes.As workload demand increases we end up with resource contention.Here we can Scale up meaning add processor with more CPU cores, add memory or use faster IO subsystem.You can scale up and it is limited by server architecture.No liner scalability |

# MPP (Massively Parallel Processing):

|  |  |
| --- | --- |
| https://cloudblogs.microsoft.com/uploads/prod/2018/03/dpi-jul30-2.jpg | Massively Parallel Processing (MPP) is the coordinated processing of a single task by multiple processors, each processor using its own OS and memory and communicating with each other using some form of messaging interface.Data is stored externally.  * Multiple CPU’s run in parallel serving single process and it has workload distribution mechanism between different nodes. * Each compute node has their own CPU, Memory and I/O subsystem. * It is **Scale Out** system where you can have multiple servers to handle the workload and you can add more compute nodes.As workload increases. So, there is no single point of contention. * It has Linear scalability. |

# Components of MPP Architecture:

|  |  |
| --- | --- |
| DistributionControl NodeCompute NodeData Movement service | Diagram  Description automatically generated |

# Distributions:

# In MPP system like SQL DW, data from each table gets divided across 60 underling distribution.

# When query is issued it is executed against each distribution in parallel.

# Distributions are stored in Azure storage

# Control Node:

# It is SQL Server End Point.

* The Control node runs the MPP engine and just stores metadata of SQL Objects and knows where the data is stored.
* Applications connect and issue T-SQL commands to a Control node, which is the single point of entry for SQL Pool.
* It break that query into parallel phases, Creates execution plan ,**co-ordinates execution of query with Compute Node**, Controls how work gets executed in parallel by compute nodes .

# Compute Node:

# Compute node is like SQL Server Engine where actual query execution takes place.

* The Compute nodes store all user data in Azure Storage and run the parallel queries.
* **Distributions are equally divided among compute nodes**
* Once the intermediate results for an individual compute node are processed, the compute nodes then return those result to the control node to be aggregate.

**Data Distribution and compute nodes**

|  |  |
| --- | --- |
| SQL DW distributions assigned to DW100 | SQL DW distributions assigned to DW600 |

# Compute Nodes can be scaled using DWUs

**(DMS)The Data Movement Service:** Itis a system-level internal service that moves data across the nodes as necessary to run queries in parallel and return accurate results.

Understanding DWU

A **DWU (**Data Warehouse Units) represents an abstract, normalized measure of compute resources and performance.

DWU = CPU, + memory + IO.

# Table Description automatically generated

**Billing in Azure Synapse Analytics:**

* Storage and compute costs are billed separately.
* Compute cost is billed based on DWU’s
* DWU’s billing is per hour.When not in use it can be paused for maximum savings.
* Storage transactions are not billed. You only pay for stored data and not storage transactions.

# Geo-redundant storage for disaster recovery is billed separate.

Refer:<https://azure.microsoft.com/en-us/pricing/details/synapse-analytics/>

**Advantages of Decoupled storage and Compute:**

* Independently size compute power irrespective of your storage needs.
* Grow or shrink compute power, within a SQL pool (data warehouse), without moving data.
* Cost saving
  + Pause compute capacity while leaving data intact, so you only pay for storage.
  + Resume compute capacity during operational hours.

## **Managing SQL DW Compute:**

## You can scale compute to meet performance demands independent of data storage as SQL Datawarehouse architecture separates storage and compute.

## Loading and query performance can increase linearly as you add more data warehouse units.

## **Scaling compute:**

* You can scale out or scale back compute by adjusting the [data warehouse units](https://docs.microsoft.com/en-us/azure/sql-data-warehouse/what-is-a-data-warehouse-unit-dwu-cdwu) (DWU)
* Scale out can be performed using
  + Azure Portal
  + PowerShell or Rest API: You can programmatically perform this operation dynamically.
  + Azure function: Used to automate the compute management operations

ALTER DATABASE AdventureworksDW

MODIFY(SERVICE\_OBJECTIVE =’DW1000’)

**Lab3: Provision Dedicated SQL Pool**

Synapse Studio🡪Manage🡪SQL Pools🡪New

Graphical user interface, text, application, email

Description automatically generated🡪Review+Create🡪Create

**Lab3: Connect to Data Warehouse using Azure Data Studio**

Copy Dedicated SQL Endpoint

Add Connection🡪 Graphical user interface, text, application, email

Description automatically generated

*Note: You can also use SSMS to connect to SQLPool.*

**Create Table**

* SQL Data Warehouse uses common create table syntax.
* Use WITH clause to specify **distribution method** to distribute data across multiple databases or distributions.
* With clause is also used to specify index, By default, clustered column store index is created on table
* **CTAS** (Create table as Select ) can also be used to create table. It is fully parallel and minimally logged operation.
* You can also create **external tables.**
* External tables is a metadata object that gets created within SQL DW that references data which is stored outside of SQL Datawarehouse. It allows you to query that external data.
* Rows are allocated to a distribution based on distribution method specified while creating table.

Example:

|  |  |
| --- | --- |
| CREATE TABLE [dbo].[FactOrder] (  [OrderKey] int NOT NULL,  [ProductKey] int NOT NULL,  [Quantity] smallint NOT NULL ) WITH­­ (  CLUSTERED COLUMNSTORE INDEX,  DISTRIBUTION = HASH([ProductKey]) )  Note:If distribution is not provided by default it is considered as ROUND ROBIN | **CTAS Statement**  CREATE TABLE [dbo].[DimProductSubcategoryCTAS]  WITH  (  DISTRIBUTION = ROUND\_ROBIN  , CLUSTERED COLUMNSTORE INDEX  )  AS  SELECT  \*  FROM [dbo].[DimProductSubcategory]  ;  Note: Distribution is compulsory |

***Note:*** *Azure SQL datawarehouse does not have same nature as SQL server as we use On-Prem.So Azure SQL DW requires design decision different than SQL Server*.

**Table Constraints:**

Azure SQL Data Warehouse supports these table constraints:

* PRIMARY KEY is only supported when NONCLUSTERED and NOT ENFORCED are both used.
* UNIQUE constraint is only supported with NOT ENFORCED is used.
* FOREIGN KEY constraint is not supported in Azure SQL Data Warehouse.

**Table Distribution Methods**

There are three distribution methods:

1. Round Robin
2. Hash
3. Replicated

**Round Robin Distributed Table:**

* A round-robin distributed table **distributes data evenly** across the table but without any further optimization.
* This is **default distribution method** if you do not specify distribution while creating table.
* You need not specify any distribution column as SQL Datawarehouse determines where the row is going to placed randomly.
* It is **quick to load data** into a round-robin table, but query performance can often be better with hash distributed tables as data having same values used for join or grouping might end up in separate distributions and can be handled by separate compute nodes.
* This delivers **fast performance** when used as a **staging table for loads.**
* Optimal for large table without good hash column or varied queries.

**Hash Distributed Table:**

* Uses deterministic hash function to assign each row to one of the 60 distributions.
* Rows are distributed based on hash value of distribution column
* Same value will always hash to same distribution.
* This can deliver the **highest query performance** for joins and aggregations on large tables. It is **optimal** for large **Fact Tables and Large Dimension tables.**
* ***Caution: Choose distribution column carefully***

|  |  |
| --- | --- |
| CREATE TABLE [dbo].[FactOrder] (  [OrderKey] int NOT NULL,  [ProductKey] int NOT NULL,  [Quantity] smallint NOT NULL ) WITH (  CLUSTERED COLUMNSTORE INDEX,  DISTRIBUTION = HASH([ProductKey]) ) | Diagram  Description automatically generated |

**Lab 4:Create table with different distributions .**

**Refer:** Distribution Demo

**Replicated Tables:**

* Replicated tables are tools for avoiding data movement for small tables
* It has distribution like Round Robin.
* It caches a full copy of the table on each compute node.
* Replicated table typically work well for small tables that are dimension tables in star schema.

**Benefits of replicated Tables:**

* Reduces data movement between compute nodes before join or aggregation
* Simplifies query plans.

|  |  |
| --- | --- |
| Diagram  Description automatically generated | **Replicated table Candidates:**  Small dimension tables in star schema  Table size <2gb  Good for small lookup tables.  Data changes infrequently, slowly changing dimension (Customer name, address etc)  Query using simple predicate -equality or inequality |

**Understanding Data Movement:**

|  |
| --- |
| Table  Description automatically generated |
| Graphical user interface, text, application  Description automatically generated |
| Table  Description automatically generated |

**Understanding Columstore**

**RowStore:**

* Rowstore index works best with the queries that seek data into data that are pulling back few rows, singleton row or range of data.

**ColumnStore :**

How does Column Store speed up queries
 

**Rowgroup:**

* For high performance and high compression rates, the columnstore index slices the table into rowgroups, and then compresses each rowgroup in a column-wise manner. The number of rows in the rowgroup must be large enough to improve compression rates.
* A rowgroup is a group of rows that are compressed into columnstore format at the same time.
* When rowgroup is compressed,each column in group is compressed.
* A rowgroup usually contains the maximum number of rows per rowgroup, which is 1,048,576 rows(2^20).
* Each column in rowgroup is compressed (called column segment)and stored separately onto physical media.
* Each rowgroup contain one column segment for every column in the table.

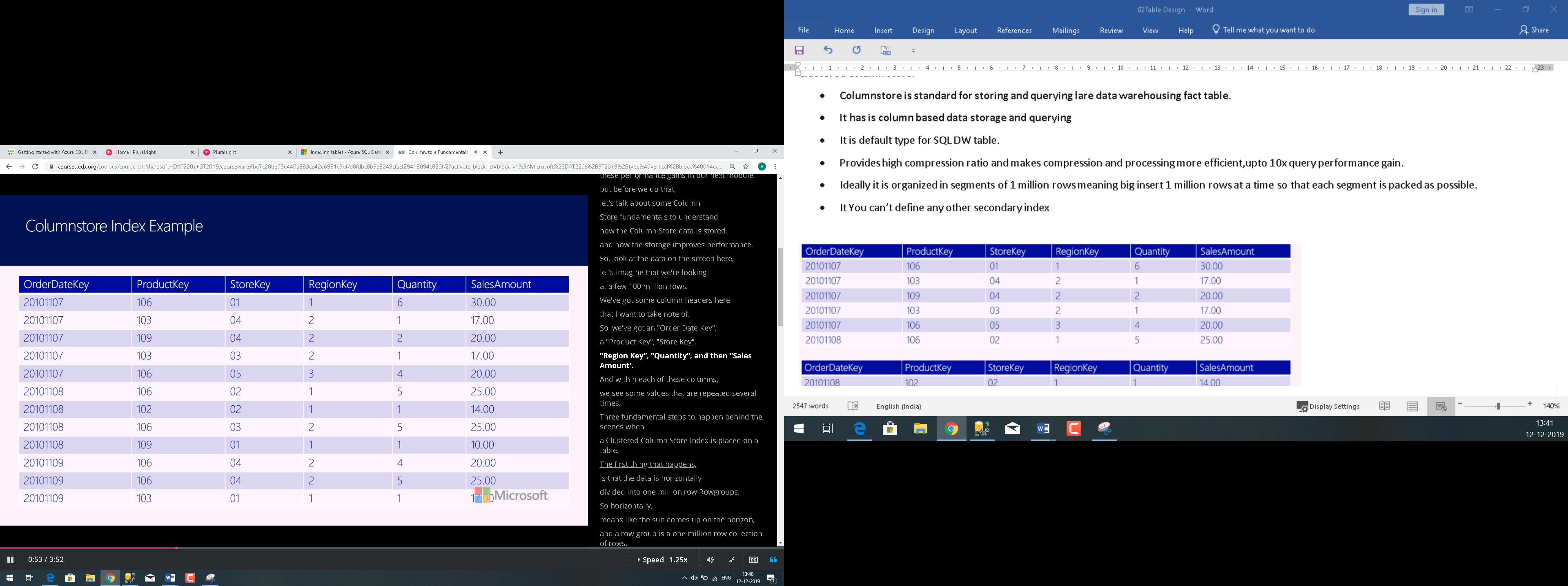
**Column Segment:**

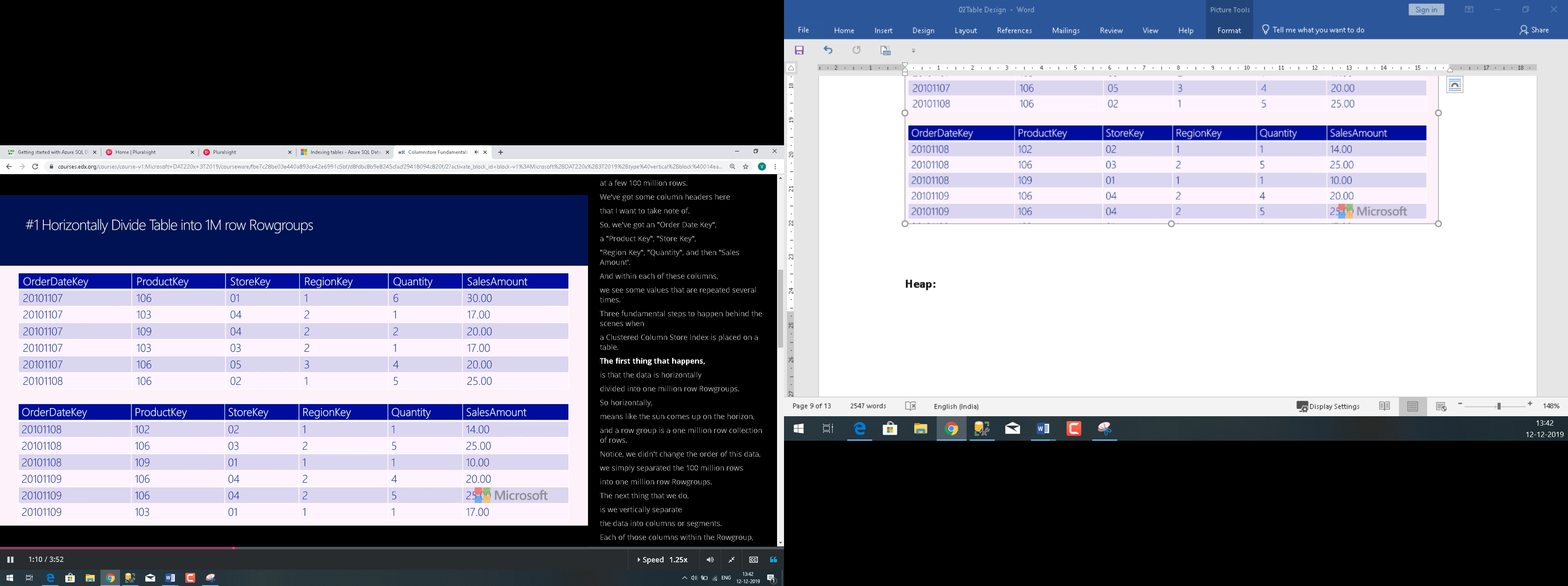
* Basic unit of storage in column store index.
* Segment contain values from one column for set of rows.
* It is group of column values which are compressed and stored together on physical media and unit of transfer between disk and memory.
* Each columnstore is comprised of one or more column segments

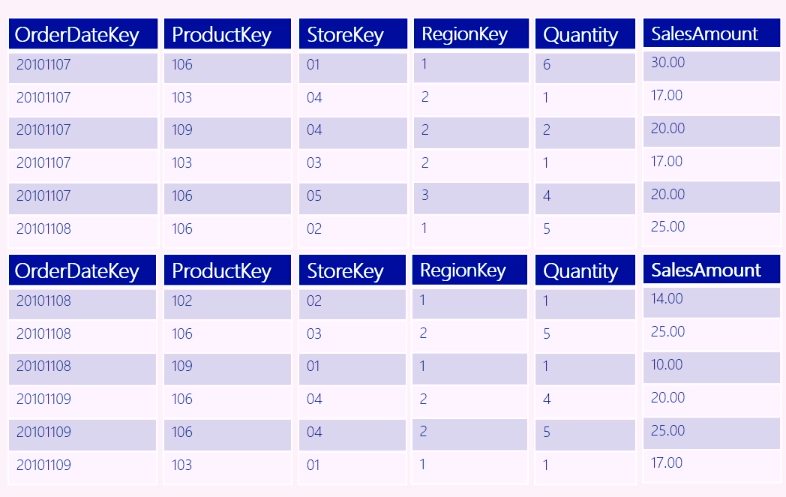
**Delta Store:**

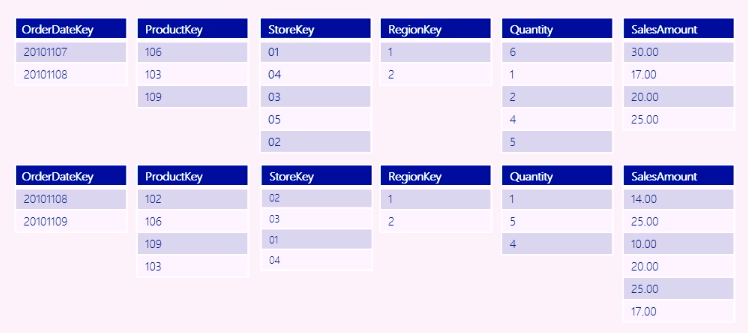
* Deltastore is rowstore that holds rows until it becomes large enough to be moved in columnstore.
* Rows are accumulated in delta store until number of rows is the maximum number of rows allowed for rowgroup

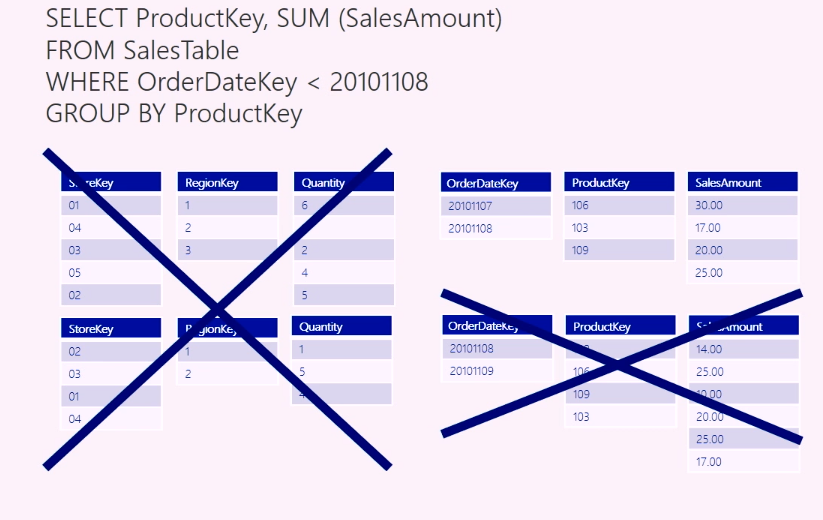
**Columstore Example:**











**Advantages of Columnstore:**

Reasons why columnstore indexes are so fast:

* Columns store values from the same domain and commonly have similar values, which result in **high compression rates**. I/O bottlenecks in your system are minimized or eliminated, and **memory footprint is reduced** significantly.
* High compression rates **improve query performance** by using a smaller in-memory footprint. In turn, query performance can improve because SQL Server can perform more query and data operations in memory.
* Batch execution improves query performance, typically by two to four times, by processing multiple rows together.
* Queries often select only a few columns from a table, which reduces total I/O from the physical media.

**Criteria for using Rowstore and columnstore:**

|  |  |
| --- | --- |
| **Rowstore**   * Small Data sets * Frequent updates * Highly selective queries * Small dimension table | **Columstore**   * Large Data sets * Mostly append only data * Analytic queries * Fact Tables or large dimension tables |

Polybase

* Polybase is technology which allows us to access data stored in external storage using T-SQL.
* It is fastest way to read and write data in external storage using T-SQL as it takes advantage of MPP Architecture.
* It supports Azure Blob storage or Azure Data Lake
* It supports different file formats like
  + Popular UTF-8 /UTF-16 delimited files
  + Popular Hadoop file formats like RC file, ORC, Parquet
  + gzip, Snappy compressed files.

**Loading Data Without Polybase:**

|  |  |
| --- | --- |
|  | SQL Pool supports many Non-Polybase methods   * BCP * SQLBulkCopy API |

**Loading Data with Polybase:**

|  |  |
| --- | --- |
| Diagram  Description automatically generated | * Polybase is fastest and most scalable way to load data. * Polybase can load data with different threads directly into compute nodes which handles the distribution. * Here control node is not involved so it scales lot better and load can happen faster. |

**Steps for setting Polybase:**

1. Create master key:It is used to encrypt credentials.
2. Create **database scoped credentials** with storage key which points to Azure blob storage.
3. Create **999 Data Source**:It references to external datasource(azure storage account)Informing polybase to use this credential to connect to path specified for blob storage.
4. Create **external file format**:to specify polybase different file format settings like delimiter etc.,which we are expecting in file
5. Create **External table**: It has metadata of underlying file and it refers to External **Data Source** and **external file format**
6. Query or load data using external table or Add external data to existing database table
7. Optimize columnstore compression and statistics.

External Table:

* PolyBase uses external tables to define and access the data in Azure Storage or Azure Data Lake Storage.
* The external table contains the table schema and points to data that is stored outside the data warehouse.
* Defining external tables involves specifying the data source, the format of the text files, and the table definitions.

*Note:*

*REJECT\_VALUE = reject\_value Specifies the value or the percentage of rows that can be rejected before the query fails.*

*For REJECT\_TYPE = value, reject\_value must be an integer between 0 and 2,147,483,647.*

*For REJECT\_TYPE = percentage, reject\_value must be a float between 0 and 100.*

**Lab5: Load data from Data Lake to SQL Pool using POLYBASE.Use Csv File**

**Refer to script:** Taxizones-ExternalTable.SQL

**Ref Docs:**

[**https://docs.microsoft.com/en-us/sql/t-sql/statements/create-external-file-format-transact-sql?view=sql-server-ver15**](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-external-file-format-transact-sql?view=sql-server-ver15)

**https://docs.microsoft.com/en-us/sql/t-sql/statements/create-external-table-transact-sql?view=sql-server-ver15**

**Ingest With Copy Statement**

* The COPY statement is the most flexible and secure way of bulk loading data in Synapse SQL.
* You can COPY data from Azure Blob Storage and Data Lake .
* It has simple syntax and allows to use wildcards and multiple files in the storage location path
* You can specify ERRORFILE Location.

**Refer:** [COPY INTO (Transact-SQL) - (Azure Synapse Analytics) - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/t-sql/statements/copy-into-transact-sql?view=azure-sqldw-latest&preserve-view=true)

[Authentication mechanisms with the COPY statement - Azure Synapse Analytics | Microsoft Docs](https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/quickstart-bulk-load-copy-tsql-examples)

**Example:**

CREATE TABLE [dbo].[Geography]

(

[GeographyID] int NOT NULL,

[ZipCodeBKey] varchar(10) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NOT NULL,

[County] varchar(50) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NULL,

[City] varchar(50) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NULL,

[State] varchar(50) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NULL,

[Country] varchar(50) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NULL,

[ZipCode] varchar(50) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NULL

)

WITH

(

DISTRIBUTION = ROUND\_ROBIN,

CLUSTERED COLUMNSTORE INDEX

);

COPY INTO [dbo].[Geography]

FROM 'https://nytaxiblob.blob.core.windows.net/2013/Geography'

WITH

(

FILE\_TYPE = 'CSV',

FIELDTERMINATOR = ',',

FIELDQUOTE = ''

)

***Note:***

*One of the advantages COPY has over PolyBase is that it supports custom column and row delimiters.*

*In Polybase, The row delimiter in delimited-text files must be either \r, \n, or \r\n. These delimiters are not user-configurable.*

**Load data into relational data Warehouse**

**Data Warehouse Load Process Considerations**

In most cases, you should implement a data warehouse load process that performs tasks in the following order:

1. Ingest the new data to be loaded into a data lake, applying pre-load cleansing or transformations as required.
2. Load the data from files into staging tables in the relational data warehouse.(Copy Statement)
3. Load the dimension tables from the dimension data in the staging tables, updating existing rows or inserting new rows and generating surrogate key values as necessary.( Combination of INSERT,UPDATE,MERGE and CREATE TABLE AS SELECT)
4. Load the fact tables from the fact data in the staging tables, looking up the appropriate surrogate keys for related dimensions.
5. Perform post-load optimization by updating indexes and table distribution statistics.

**Create and Load staging tables**

There can be two approaches

1. Use an external table to query files in the data lake and use CTAS
2. Ingest data into staging table using COPY command

**Load Dimension Table**

You can use combination of INSERT,UPDATE,MERGE and CREATE TABLE AS SELECT to load staged data into dimension and fact tables

You can use following approaches:

1. **Use CTAS Expression to** creates a new table based on the results of a SELECT statement.

Note:You can't use IDENTITY to generate a unique integer value for the surrogate key

Load a combination of new and updated data into a dimension table by using a CREATE TABLE AS (CTAS),to create a new table that UNIONs the existing rows from the dimension table with the new and updated records from the staging table.

After creating the new table, you can delete or rename the current dimension table, and rename the new table to replace it.

1. **Use Insert statement**

When you need to load staged data into an existing dimension table, you can use an INSERT statement. This approach works if the staged data contains only records for new dimension entities

Note:You can use Identity column wih this approach.

**Load slowly Changing dimensions**

Logic to implement Type 1 and Type 2 updates can be complex, and there are various techniques you can use.

1. Combining INSERT AND UPDATE statements
2. Using Merge statement

**Lab :Load DimProduct and DimCustomer dimensions using files present in data lake.**

**Required Dataset:Customer.csv,Product.csv,Customer1.csv**

1. Ingest customer and product data into StageCustomer and StageProduct table respectively

**Refer:**PopulateStaging Table

1. Load DimProduct Table

**Refer:**LoadProductDimension

1. Populate SCD DimCustomer

**Refer:**LoadSCDCustomer

1. You can use Time Flags or Boolean flag in case of SCD2

**Refer:**SCDType2IsActive

**Load Time Dimension**

Time dimension tables store a record for each time interval based on the grain by which you want to aggregate data over time

**Example:**

-- Create a temporary table for the dates we need

CREATE TABLE #TmpStageDate (DateVal DATE NOT NULL)

-- Populate the temp table with a range of dates

DECLARE @StartDate DATE

DECLARE @EndDate DATE

SET @StartDate = '2019-01-01'

SET @EndDate = '2023-12-31'

DECLARE @LoopDate = @StartDate

WHILE @LoopDate <= @EndDate

BEGIN

INSERT INTO #TmpStageDate VALUES

(

@LoopDate

)

SET @LoopDate = DATEADD(dd, 1, @LoopDate)

END

-- Insert the dates and calculated attributes into the dimension table

INSERT INTO dbo.DimDate

SELECT CAST(CONVERT(VARCHAR(8), DateVal, 112) as INT), -- date key

DateVal, --date alt key

Day(DateVal) -- day number of month

--, other derived temporal fields as required

FROM #TmpStageDate

GO

--Drop temporary table

**DROP TABLE #TmpStageDate**

**Load Fact Tables**

Typically,regular data warehouse load operation loads fact tables after dimension tables. This approach ensures that the dimensions to which the facts will be related are already present in the data warehouse.

The staged fact data usually includes the business (alternate) keys for the related dimensions, so your logic to load the data must look up the corresponding surrogate keys

You need to identify correct version of matching row in case of SCD.

Following approach can be used

* Use an INSERT statement
* Look up surrogate keys in dimension tables based on alternate key:
  + - Simple case, get the most recently loaded dimension instance (maximum incrementing surrogate key)
    - Or use an *IsCurrent* flag field
    - Or use *start* and *end* dates to find the right instance for the fact time

Example:

INSERT INTO dbo.FactSales

SELECT (SELECT MAX(DateKey)

FROM dbo.DimDate

WHERE FullDateAlternateKey = stg.OrderDate) AS OrderDateKey,

(SELECT MAX(CustomerKey)

FROM dbo.DimCustomer

WHERE CustomerAlternateKey = stg.CustNo) AS CustomerKey,

(SELECT MAX(ProductKey)

FROM dbo.DimProduct

WHERE ProductAlternateKey = stg.ProductID) AS ProductKey,

(SELECT MAX(StoreKey)

FROM dbo.DimStore

WHERE StoreAlternateKey = stg.StoreID) AS StoreKey,

OrderNumber,

OrderLineItem,

OrderQuantity,

UnitPrice,

Discount,

Tax,

SalesAmount

FROM dbo.StageSales AS stg