Microsoft

Azure Data Warehouse In-A-Day

Steve Young - Data & AI CSA
Rebecca Young - Data & AI CSA

Agenda

Agenda:

Time	Topic	Description	Materials
09:00am - 09:15am	Introductions & Logistics (15min)	Welcome	N/A
09:15am - 10:00am	Datawarehouse Patterns in Azure & SQL DW Overview (45min)	Slide Deck 01	N/A
10:00am - 10:45am	SQL DW Gen2 New Features & Planning Your Project Build (45min)	Slide Deck 02	N/A
10:45am - 11:00pm	Break (15min)	Please take a break	N/A
11:00am -12:00pm	Demo & Lab 01 (60 Min)	Setting up the LAB environment	Lab 01
12:00pm -1:00pm	Lunch (60 Min)	Lunch and complete lab 01	N/A
01:00pm -1:30pm	SQLDW Loading Best Practices (30 Min)	Lecture	N/A
01:30pm -02:15pm	Lab 02/03: User IDs & Data loading scenarios and best practices (45min)	Loading different scenarios	Lab 02/03
02:15am - 2:30pm	Break (15min)	Please take a break	N/A
02:30pm -3:00pm	SQLDW Operational Best Practices (30 Min)	Lecture	N/A
03:00pm -03:45pm	Lab 04: Performance Tuning best practices (45min)		Lab 04
03:45pm -4:15pm	Lab 05: Lab 3: Monitoring, Maintenance and Security (30min)		Lab 05
4:15pm -5:00pm	Q&A and Wrap-up (45min)	final remarks or takeaways/next steps	Survey

Azure SQL Data Warehouse Gen 2 - Performance Features

Data Movement

https://azure.microsoft.com/en-us/blog/smarter-faster-safer-azure-sql-data-warehouse-is-simply-unmatched/

Data Movement - Instant Data Movement (IDM)

Instant Data Movement (IDM) - extremely efficient movement between data warehouse compute nodes.

At the heart of every distributed database system is the **need to align two or more tables**

That are partitioned on a different key to produce a final or intermediate result set.

Distributed Data Movement

ProductSales

	AccountID	SalesAmt	…
Node 1:	47	$1,234.36	…
Node 2:	36	$2,345.47	…
Node 3:	14	$3,456.58	…
Node 4:	25	$4,567.69	…
Node 5:	48	$5,678.70	…
Node 6:	37	$6,789.81	…
	…	…	…

SalesAccountTerritory

SATerritoryID	AccountID	…
444	37	…
333	25	
111	36	…
222	47	…
445	14	
334	48	…
…	…	…

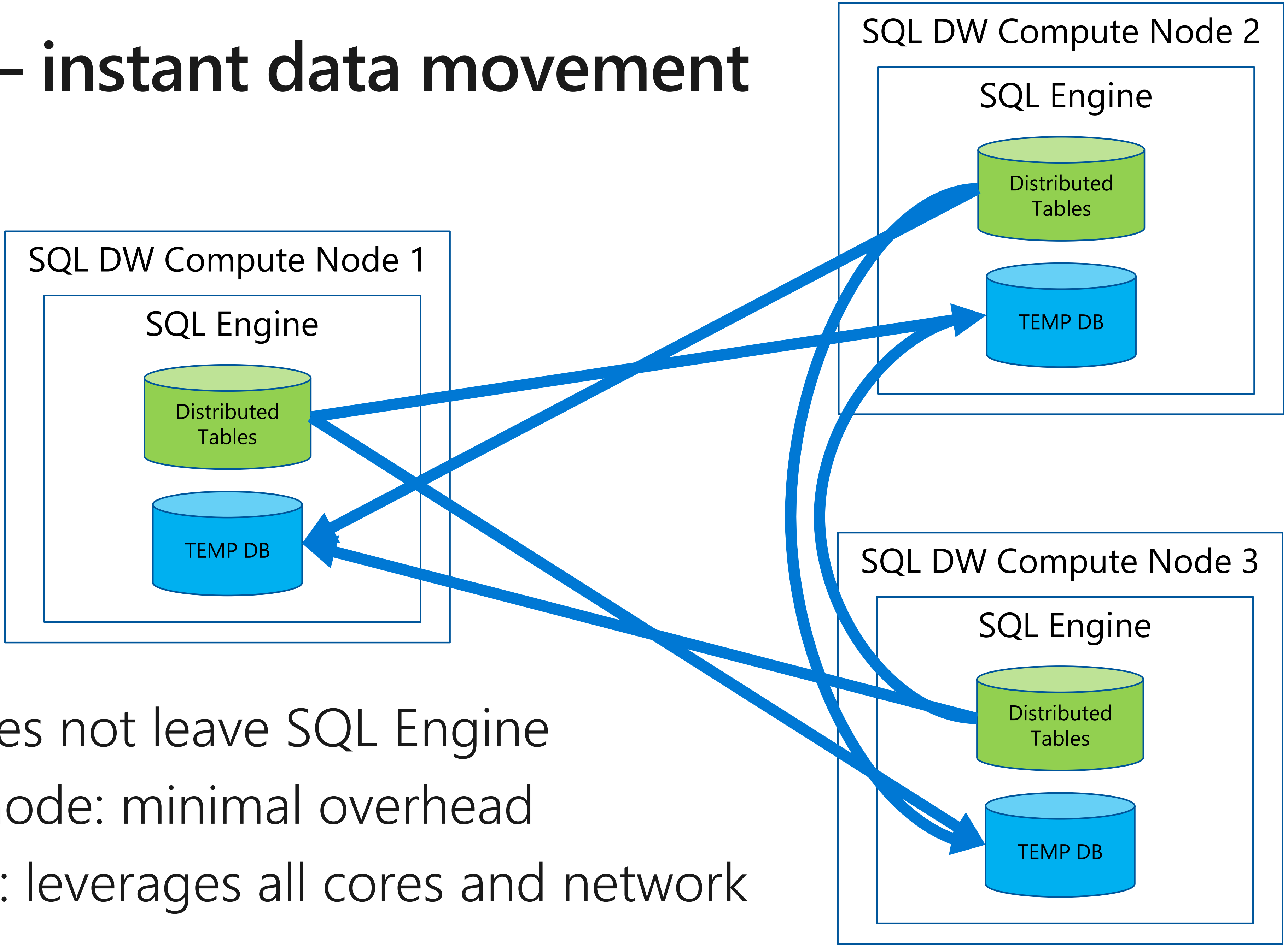
```sql
CREATE TABLE ProductSales
WITH (DISTRIBUTION=HASH(AccountID))
AS...

CREATE TABLE SalesAccountTerritory
WITH (DISTRIBUTION=HASH(SATerritoryID))
AS...

SELECT TOP 25 a.SalesAccountTerritoryName
      ,TotalSales = SUM(p.SalesAmt)
FROM ProductSales p
JOIN SalesAccountTerritory a
ON   a.AccountID = p.AccountID
GROUP BY a.SalesAccountTerritoryName
ORDER BY 2 DESC
```

Shuffle

SATName	TotalSales
North	$6,789.81
South	$5,678.70
NorthEast	$4,567.69
SouthWest	$3,456.58
East	$2,345.47
West	$1,234.36

	SATName	…
	West	…
	East	
	SouthWest	…
	NorthEast	…
	South	
37	North	…
…	…	…

Gen2 – instant data movement



Data does not leave SQL Engine

Batch-mode: minimal overhead

Scalable: leverages all cores and network

Distribution Column

Choose a distribution column that minimizes data movement

To get the correct query result queries might move data from one Compute node to another. Data movement commonly happens when queries have joins and aggregations on distributed tables. Choosing a distribution column that helps minimize data movement is one of the most important strategies for optimizing performance of your SQL Data Warehouse.

To minimize data movement, select a distribution column that:

- Is used in `JOIN`, `GROUP BY`, `DISTINCT`, `OVER`, and `HAVING` clauses. When two large fact tables have frequent joins, query performance improves when you distribute both tables on one of the join columns. When a table is not used in joins, consider distributing the table on a column that is frequently in the `GROUP BY` clause.
- Is *not* used in `WHERE` clauses. This could narrow the query to not run on all the distributions.
- Is *not* a date column. WHERE clauses often filter by date. When this happens, all the processing could run on only a few distributions.

What to do when none of the columns are a good distribution column

If none of your columns have enough distinct values for a distribution column, you can create a new column as a composite of one or more values. To avoid data movement during query execution, use the composite distribution column as a join column in queries.

Once you design a hash-distributed table, the next step is to load data into the table. For loading guidance, see Loading overview.

Resource Classes

Gen2 – Simpler resource model

With Gen2, dynamic resource pools were introduced with a 3-10-22-70 model for resource allocations.

Resource Class	Percent Resources	Concurrency
SmallRc	3%	32
MediumRc	10%	10
LargeRc	22%	4
XLargeRc	70%	1

```
EXEC sp_addrolemember 'largerc', 'loaduser';
```

Gen2 – Simpler model

At MediumRc, an example…

MediumRC		Gen1 Model		Gen2 Model	
Service Level	Total Slots	Slots	Concurrency	Slots	Concurrency
DW1000c	40	8	5	4	10
DW1500c	60	8	7	6	10
DW2000c	80	16	5	8	10
DW2500c	100	16	6	10	10
DW3000c	120	16	7	12	10
DW5000c	200	32	6	20	10
DW6000c	240	32	7	24	10
DW7500c	300	64	4	30	10
DW10000c	400	64	6	40	10
DW15000c	600	64	9	60	10
DW30000c	1200	64	18	120	10

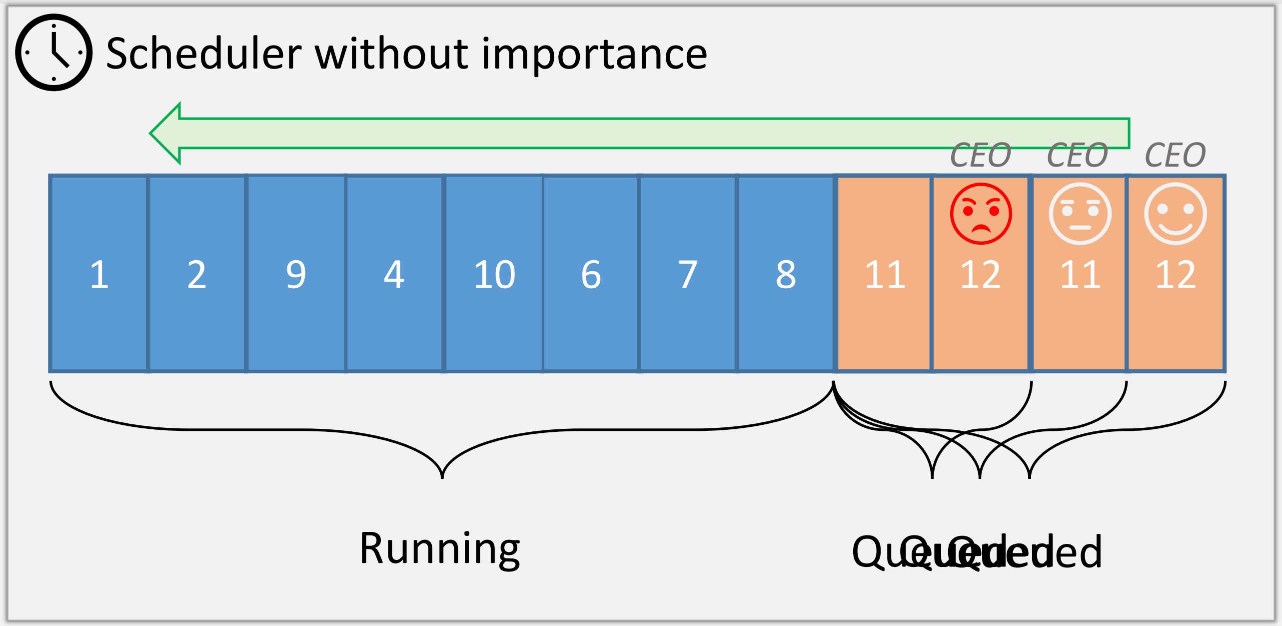
Workload Management

WORKLOAD IMPORTANCE – NO IMPORTANCE

Overview

Workload importance allows you to prioritize the queries that get access to resources.

It helps ensure that high-business value work is executed first on a busy data warehouse.



WORKLOAD CLASSIFICATION

Overview

Allows you to map a query to an allocation of resources via pre-determined rules

Use this in combination with workload importance to effectively share resources across different workload types

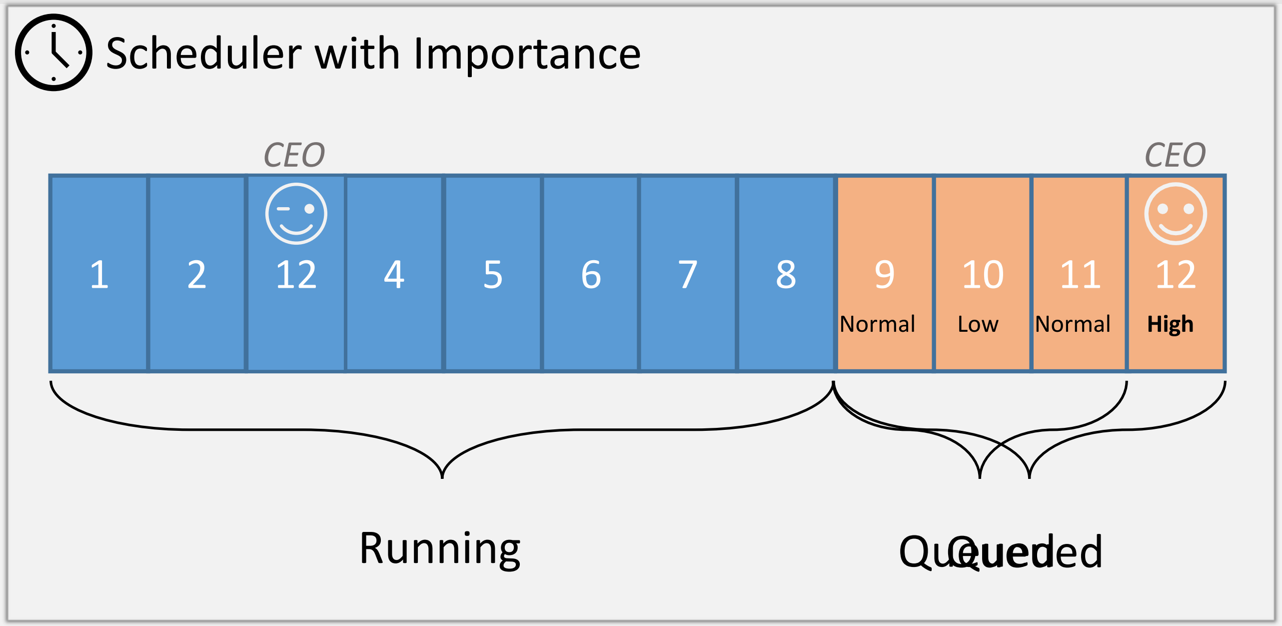
```
CREATE WORKLOAD CLASSIFIER classifier_name
WITH
(
    [WORKLOAD_GROUP = '<Resource Class>' ]
    [IMPORTANCE = {    LOW           |
                       BELOW_NORMAL  |
                       NORMAL        |
                       ABOVE_NORMAL  |
                       HIGH
                  }
    ]
    [MEMBERNAME = ‘security_account’]
)
```

WORKLOAD IMPORTANCE –IMPORTANCE

Overview

Workload importance allows you to prioritize the queries that get access to resources.

It helps ensure that high-business value work is executed first on a busy data warehouse.



WORKLOAD ISOLATION (PREVIEW)

Overview

Isolation allocates fixed resources to workloads within a data warehouse. These limits are strictly enforced for memory, and only enforced under load for CPU and IO.

Resource classes are implemented by assigning users to database roles. When a user runs a query, the query runs with the user's resource class

EXEC sp_addrolemember 'largerc', 'loaduser';

EXEC sp_droprolemember 'largerc', 'loaduser';

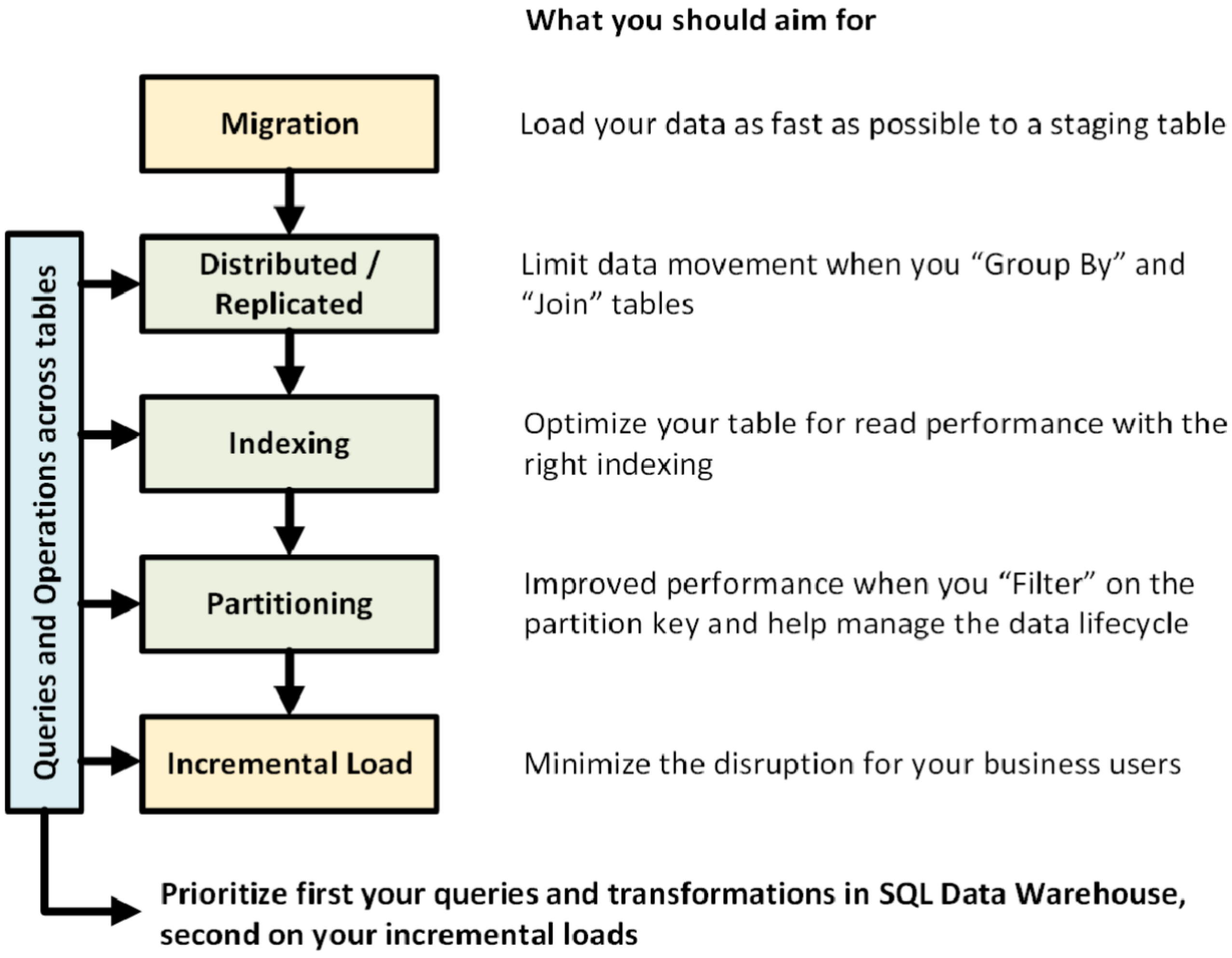
```
CREATE WORKLOAD GROUP group_name
WITH
(
    [ MIN_PERCENTAGE_RESOURCE = value ]
    [ CAP_PERCENTAGE_RESOURCE = value ]
    [ MAX_CONCURRENCY = value ]
)
```

https://docs.microsoft.com/en-us/azure/sql-data-warehouse/sql-data-warehouse-workload-classification

DW Creation Framework Architecture Decisions

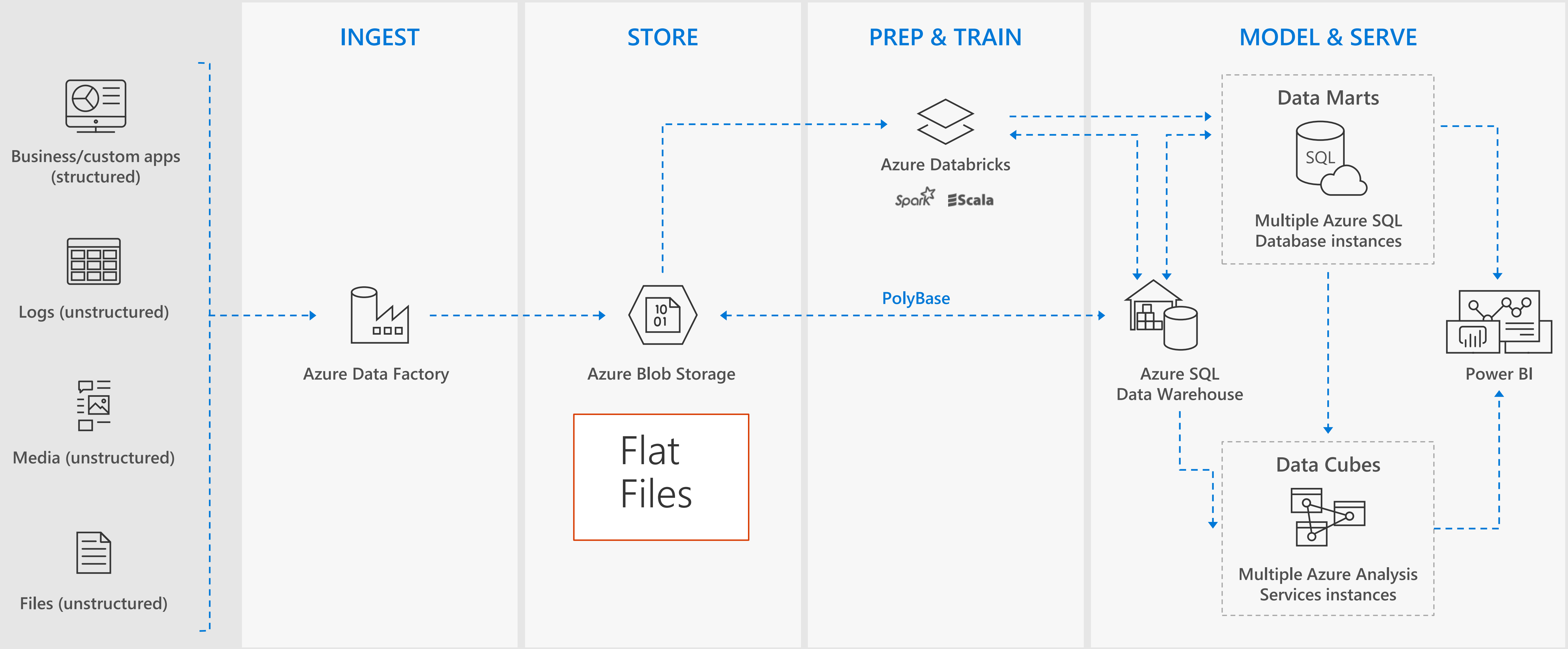
DW Creation Framework Architecture Decisions

"Improve your probability of Success!!!!!"



https://docs.microsoft.com/en-us/azure/sql-data-warehouse/cheat-sheet

Hub & Spoke Architecture for Analytics (BI)



Microsoft Azure supports other services like Azure HDInsight and Azure Data Lake in various layers to allow customers a truly customized solution.

Choose a Light House Project

Don't let your first project be your last !!!

- Get Comfortable with the technology – Kick the tires
- Take a smaller project, treat like a production roll out.
- Provides insights not only about the technology, but the organization's maturity in being able to do a data project
  - Can the Input Files be generated, Tested, Errors traced
  - Handle change management
  - If you don't measure, how do you know if you succeeded? Failed?

DW Creation Framework

Data Warehouse Workflow

Inspire

Imagine a future state solution and develop use cases and query patterns

Solution Design

Design - Sources

Discovery the source data and plan for ELT/ETL Process

Identify Data Process

Design SQL Architecture

Empower others in your organization to see the value

Solution Demonstration

Operate

Achieve business outcomes and improved customer experiences

Operation

DATA WAREHOUSE DESIGN WORKFLOW

Inspire - Project Goals and Definitions

Data Architecture and Flow	Queries and Table Joins	Security / Access Users / Resource Classes

Design - Identify Data Sources

Identify Data Migration techniques	Incremental Load	Migration Pipelines

Design - SQL DW Tables

External Table Definitions	Staging Tables	Production Tables	Indexing Patterns	Partitioning	Distribution and Replicated

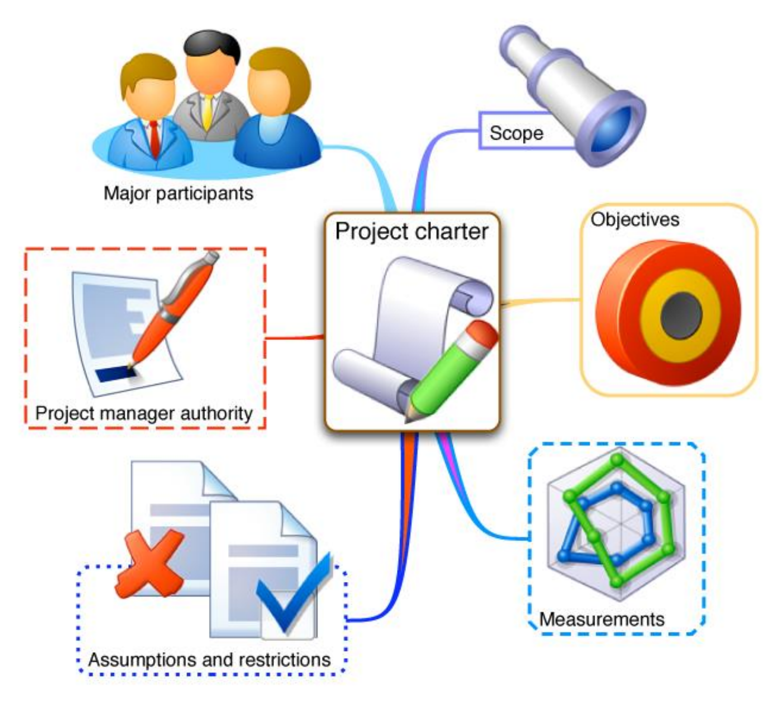
Operations

Indexing Patterns	Query Performance

WORKFLOW – PROJECT GOALS AND DEFINITIONS

Imagine a future state solution and develop use cases and query patterns

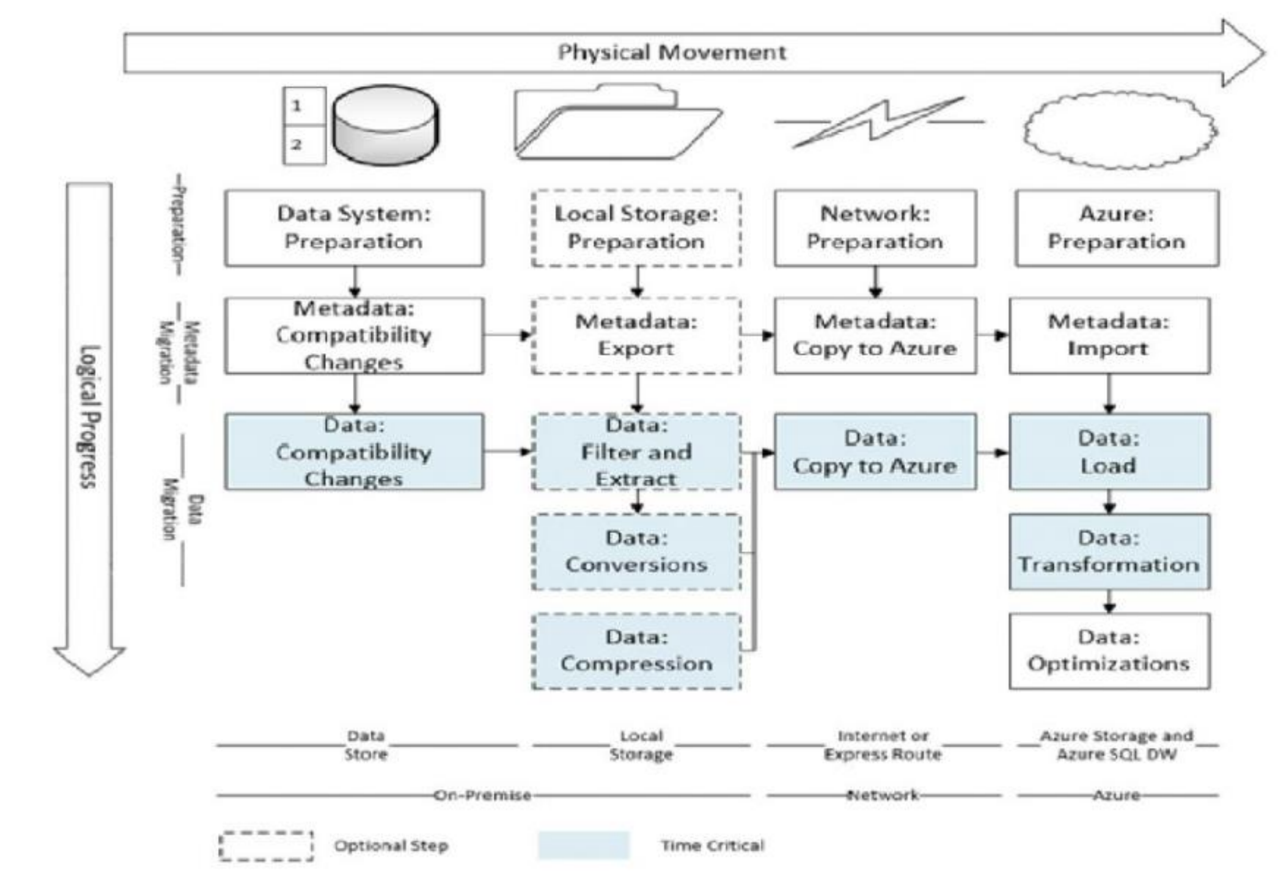
- Data Architecture and Flow - Identification
  - What sources are to be used
  - What format should they take
  - What is the update pattern
- Queries and Table Joins
  - Capture existing queries, reports and ad-hoc analysis
- Security / Access Users / Priorities (Resource Classes)



WORKFLOW – IDENTIFY DATA SOURCES

Imagine a future state solution and develop use cases and query patterns

- Identify Data Migration tasks
    - File formats
    - Update schedule
- Incremental Load
    - Are these required?
- Migration Pipelines
    - Azure Data Factory
    - Azure Data Lake / Blob storage



https://blogs.msdn.microsoft.com/sqlcat/2016/08/18/migrating-data-to-azure-sql-data-warehouse-in-practice/

https://techcommunity.microsoft.com/t5/DataCAT/Azure-SQL-Data-Warehouse-loading-patterns-and-strategies/ba-p/305456

WORKFLOW – IDENTIFY DATA SOURCES

Imagine a future state solution and develop use cases and query patterns

Using CTAS to load initial data

Then you can use a CTAS (CREATE TABLE AS SELECT) operation within SQL Data Warehouse to load the data from Azure Blob Storage to SQL Data Warehouse:

```
CREATE TABLE orders_load
WITH (CLUSTERED COLUMNSTORE INDEX, DISTRIBUTION = HASH(o_orderkey),
     PARTITION (o_orderdate RANGE RIGHT FOR VALUES ('1992-01-01','1993-01-01','1994-01-01','1995-01-01')))
 as select * from orders_ext;
```

CTAS creates a new table. We recommend using CTAS for the initial data load. This is an all-or-nothing operation with minimal logging.

Using INSERT INTO to load incremental data

For an incremental load, use INSERT INTO operation. This is a full logging operation when inserting into a populated partition which will impact on the load performance. Furthermore, the roll-back operation on a large transaction can be expensive. Consider breaking your transaction into smaller batches.

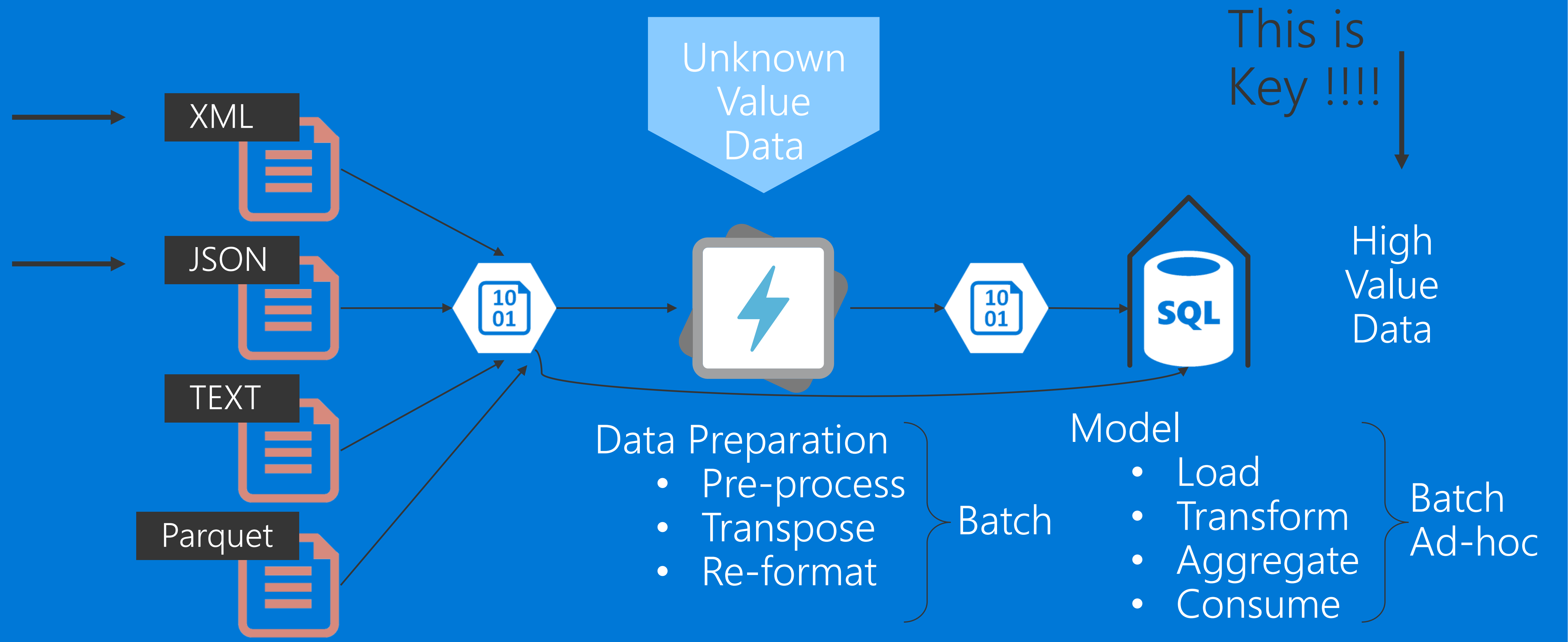
```
INSERT INTO TABLE orders_load
select * from orders_current_ext;
```

Note The source is using different external table, orders_current_ext. This is the external table defining the path for the incremental data on ASB.

Another popular pattern is to load into a partitioned aligned stage table via CTAS, then partition switch into the final table.

https://techcommunity.microsoft.com/t5/DataCAT/Azure-SQL-Data-Warehouse-loading-patterns-and-strategies/ba-p/305456

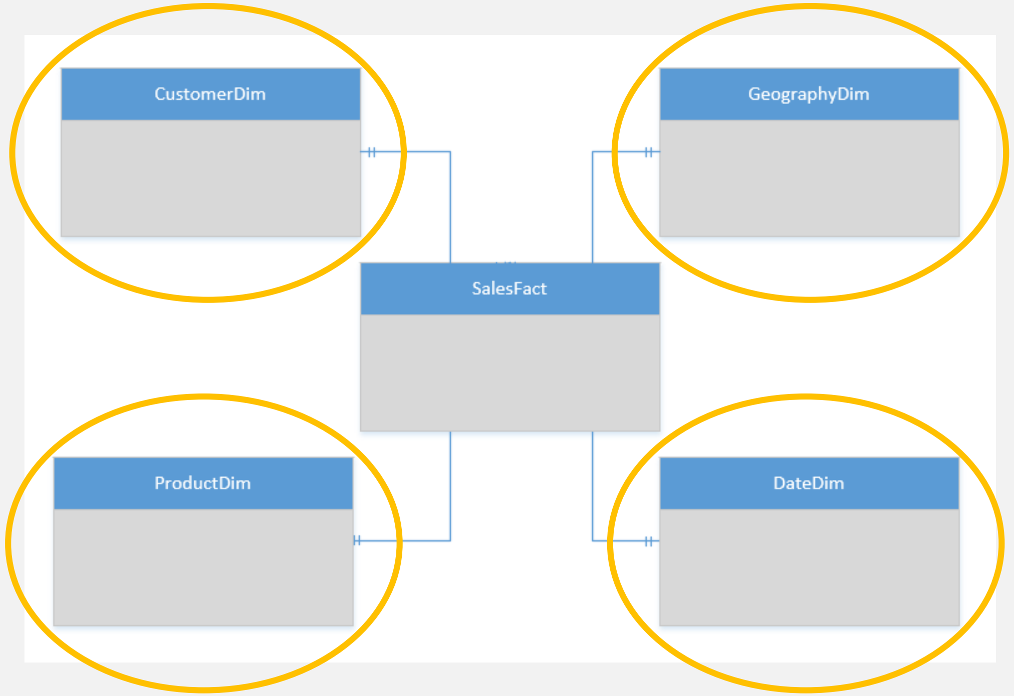
WORKFLOW – IDENTIFY DATA SOURCES



WORKFLOW – DESIGN SQL DW TABLES

Imagine a future state solution and develop use cases and query patterns

- External Table Definitions
- Staging Tables
- Production Tables
- Indexing Patterns
- Partitioning
- Distribution and Replicated




```sql
-- Create a database master key if one does not already exist, using your own password. This key is used to encrypt the
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'S0me!nfo'
;

-- Create a database scoped credential with Azure storage account key as the secret.
CREATE DATABASE SCOPED CREDENTIAL AzureStorageCredential
WITH
     IDENTITY   = '<my_account>'
,    SECRET     = '<azure_storage_account_key>'
;

-- Create an external data source with CREDENTIAL option.
CREATE EXTERNAL DATA SOURCE MyAzureStorage
WITH
(    LOCATION   = 'wasbs://daily@logs.blob.core.windows.net/'
,    CREDENTIAL = AzureStorageCredential
,    TYPE       = HADOOP
)
;
```

```sql
CREATE TABLE dbo.DimCustomer
(    CustomerKey            int            NOT NULL
,    GeographyKey           int            NULL
,    CustomerAlternateKey   nvarchar(15)   NOT NULL
,    Title                  nvarchar(8)    NULL
,    FirstName              nvarchar(50)   NULL
,    LastName               nvarchar(50)   NULL
,    BirthDate              date           NULL
,    Gender                 nvarchar(1)    NULL
,    EmailAddress           nvarchar(50)   NULL
,    YearlyIncome           money          NULL
,    DateFirstPurchase      date           NULL
)
WITH
(    CLUSTERED COLUMNSTORE INDEX
,    DISTRIBUTION = REPLICATED
)
```

WORKFLOW – OPERATIONS

Imagine a future state solution and develop use cases and query patterns

- Indexing Patterns
- Query Performance
- Data Management Views

Microsoft

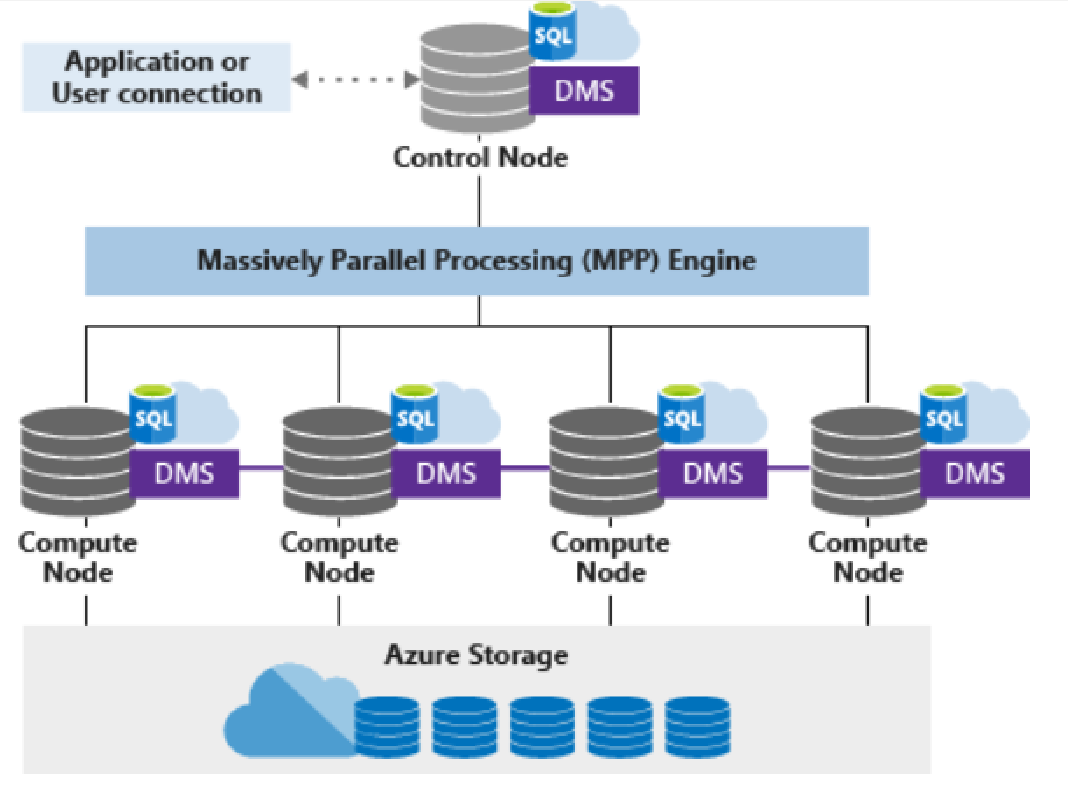
Q&A

SQL Datawarehouse Architecture

ARCHITECTURE - OVERVIEW

Decoupled Storage allows:

- Independently size compute power irrespective of your storage needs.
- Grow or shrink compute power without moving data.
- Pause compute capacity while leaving data intact, so you only pay for storage.
- Resume compute capacity during operational hours.

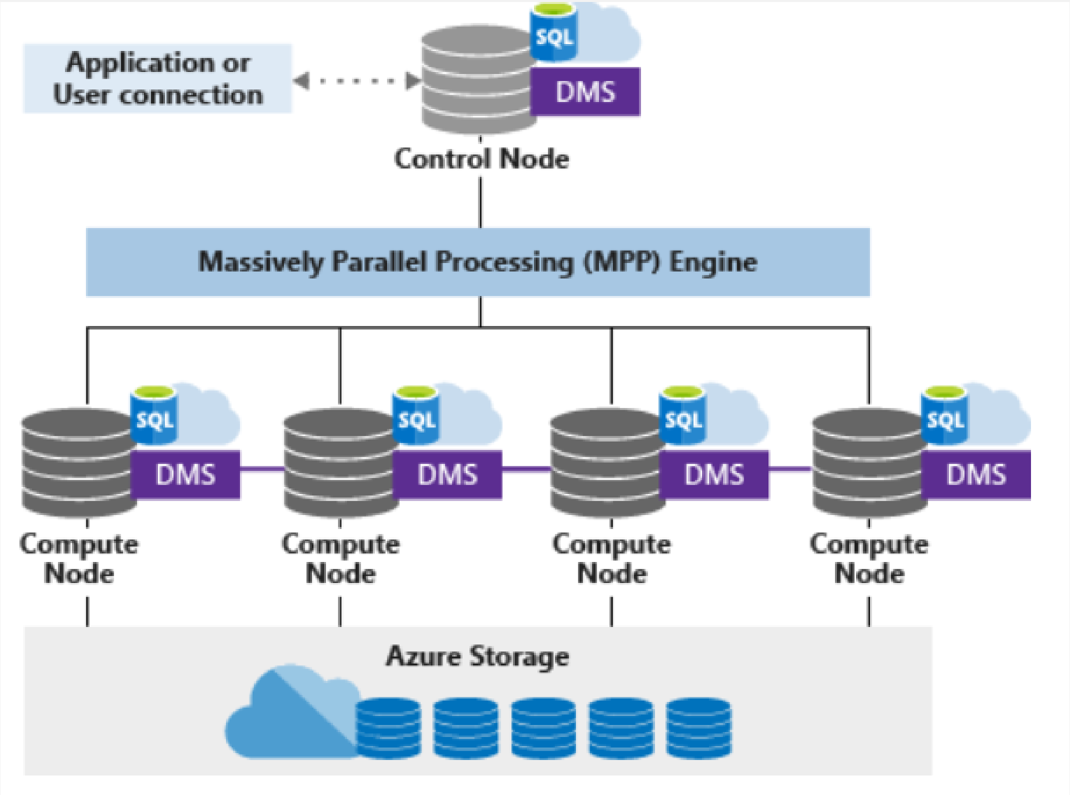


[Reference - https://docs.microsoft.com/en-us/azure/sql-data-warehouse/massively-parallel-processing-mpp-architecture](https://docs.microsoft.com/en-us/azure/sql-data-warehouse/massively-parallel-processing-mpp-architecture)

ARCHITECTURE - OVERVIEW

Overview

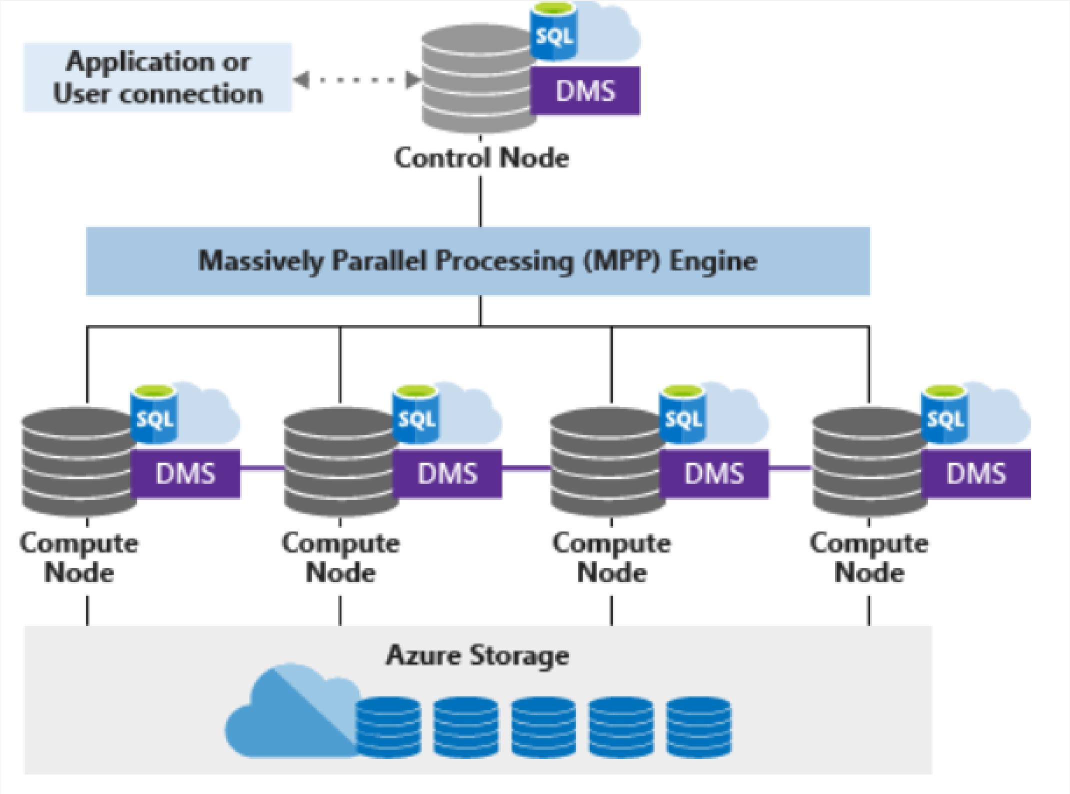
- SQL Data Warehouse uses a node-based architecture.
- Applications connect and issue T-SQL commands to a Control node, single point of entry
- The Control node runs the MPP engine which optimizes queries for parallel processing, and then passes operations to Compute nodes to do their work in parallel.
- The Compute nodes store all user data in Azure Storage and run the parallel queries.
- The Data Movement Service (DMS) moves data across the nodes as necessary to run queries in parallel and return accurate results.



ARCHITECTURE – AZURE STORAGE

Overview

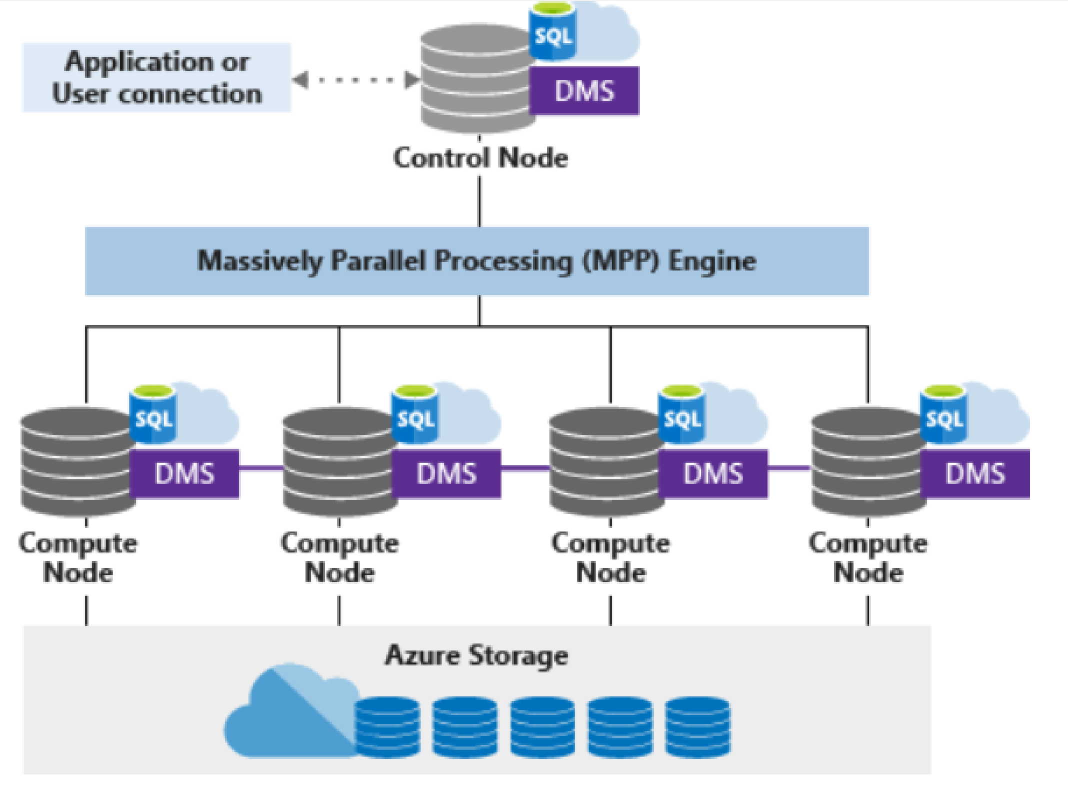
- Data is stored and managed by Azure storage
- The data itself is sharded into distributions to optimize the performance of the system. You can choose which sharding pattern to use to distribute the data when you define the table. SQL Data Warehouse supports these sharding patterns:
    - Hash
    - Round Robin
    - Replicate



ARCHITECTURE – CONTROL NODE

Overview

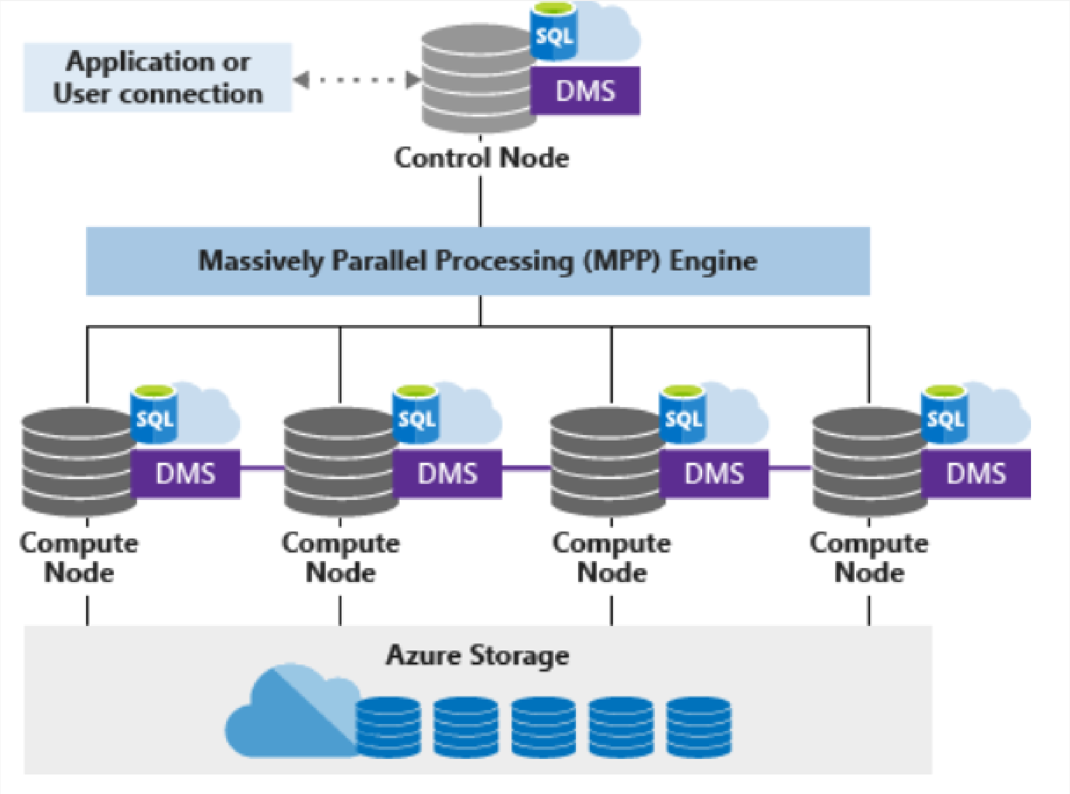
- The brain of the data warehouse
- The front end that interacts with all applications and connections.
- MPP engine runs on the Control node to optimize and coordinate parallel queries.
- The Control node transforms a submit a T-SQL query into queries that run against each distribution in parallel.



ARCHITECTURE – COMPUTE NODES

Overview

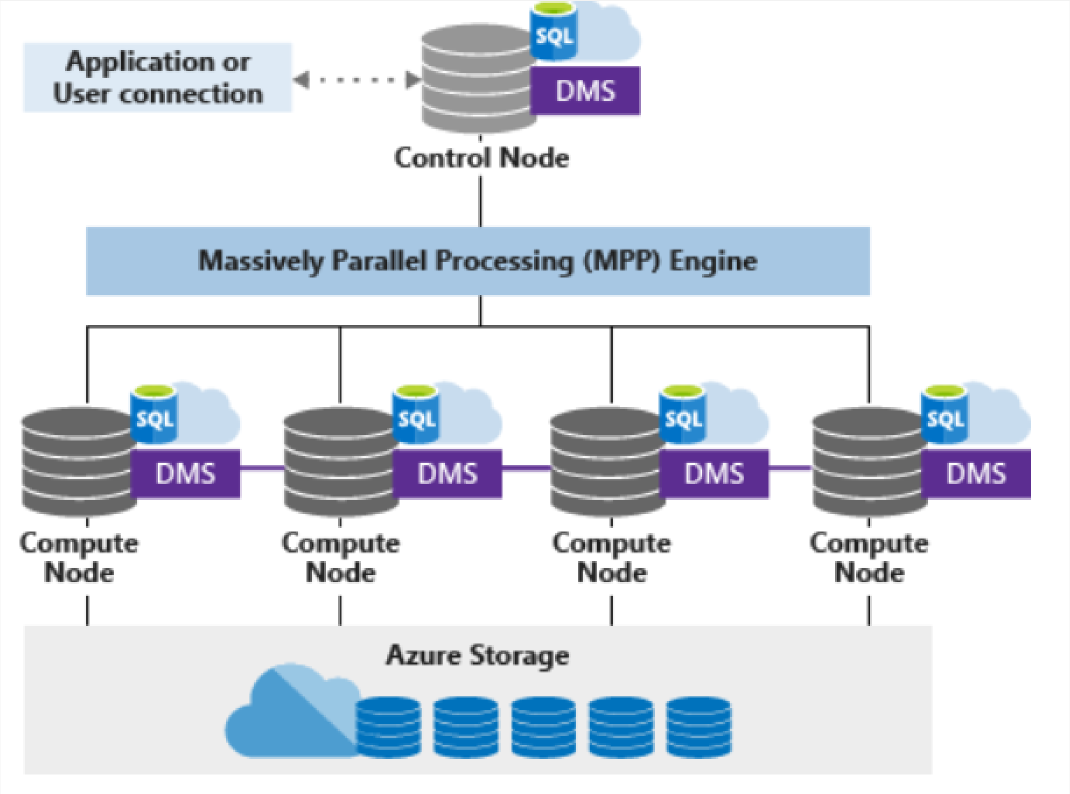
- Provide the computational power.
- Distributions map to Compute nodes for processing.
- Scale Up compute resources, SQL Data Warehouse re-maps the distributions to the available Compute nodes which ranges from 1 to 60
- Each Compute node has a node ID that is visible in system views. https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sql-data-warehouse-and-parallel-data-warehouse-catalog-views?view=aps-pdw-2016-au7



ARCHITECTURE – DATA MOVEMENT SERVICE

Overview

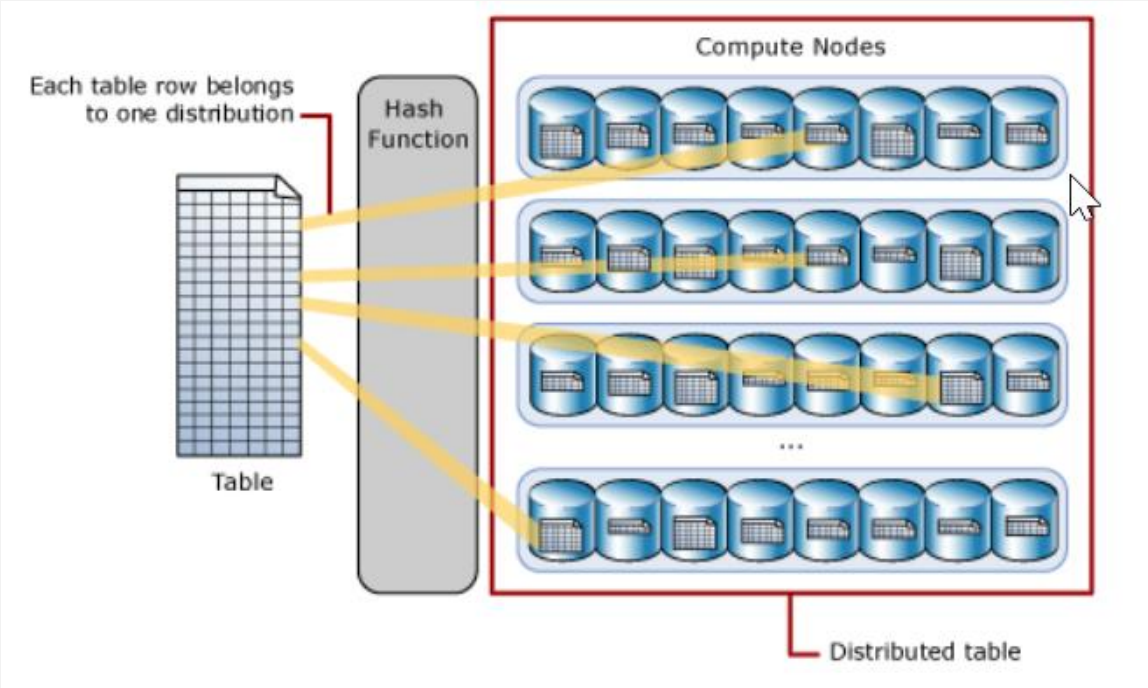
- The data transport technology that coordinates data movement between the Compute nodes.
- Some queries require data movement to ensure the parallel queries return accurate results.



ARCHITECTURE – TABLE DISTRIBUTIONS

Hash Table Distributions

- A hash distributed table can deliver the highest query performance for joins and aggregations on large tables.
- Uses a hash function to deterministically assign each row to one distribution.
- In the table definition, one of the columns is designated as the distribution column.
- The hash function uses the values in the distribution column to assign each row to a distribution.



ARCHITECTURE – TABLE DISTRIBUTIONS

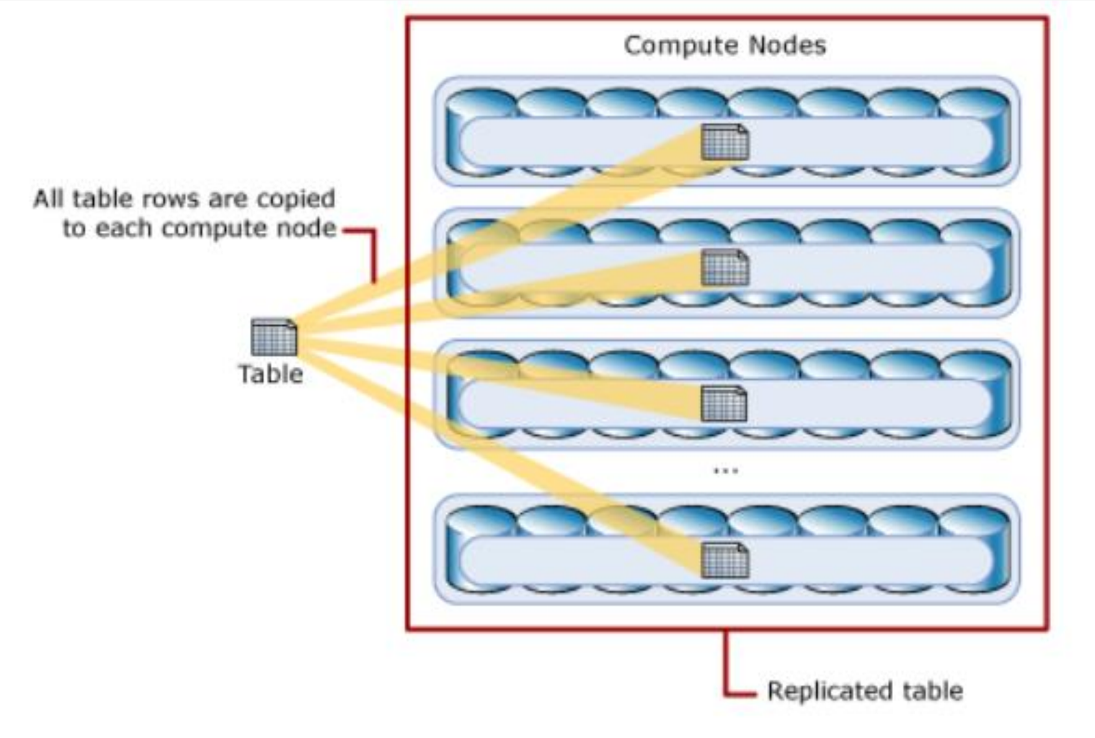
Round-robin distributed Table Distributions

- Round-robin table is the simplest table to create and delivers fast performance when used as a staging table for loads.
- A round-robin distributed table distributes data evenly across the table but without any further optimization.
- Query performance can often be better with hash distributed tables.
- Joins on round-robin tables require reshuffling data and this takes additional time.

ARCHITECTURE – TABLE DISTRIBUTIONS

Round-robin distributed Table Distributions

- A replicated table provides the fastest query performance for small tables.
- A table that is replicated caches a full copy of the table on each compute node. Consequently, replicating a table removes the need to transfer data among compute nodes before a join or aggregation. Replicated tables are best utilized with small tables. Extra storage is required and there is additional overhead that is incurred when writing data which make large tables impractical.



Performance Details Resources

Gen2 GA

Turbocharge cloud analytics with Azure SQL Data Warehouse

Blazing fast data warehousing with Azure SQL Data Warehouse

Adaptive Caching

Adaptive caching powers Azure SQL Data Warehouse performance gains

Instant data movement

Lightning fast query performance with Azure SQL Data Warehouse

GigaOm Benchmarking Report

https://gigaom.com/report/data-warehouse-in-the-cloud-benchmark/

Result-set caching

Overview

Cache the results of a query in DW storage. This enables interactive response times for repetitive queries against tables with infrequent data changes.

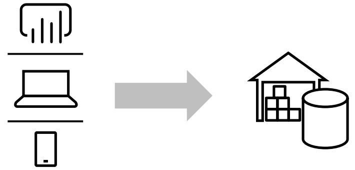
The result-set cache persists even if a data warehouse is paused and resumed later.

Query cache is invalidated and refreshed when underlying table data or query code changes.

Result cache is evicted regularly based on a time-aware least recently used algorithm (TLRU).

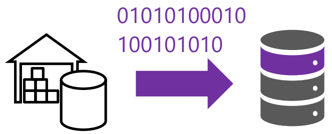
```
-- Turn on/off result-set caching for a database
-- Must be run on the MASTER database
ALTER DATABASE {database_name}
SET RESULT_SET_CACHING { ON | OFF }

-- Turn on/off result-set caching for a client
session
-- Run on target data warehouse
SET RESULT_SET_CACHING {ON | OFF}

-- Check result-set caching setting for a database
-- Run on target data warehouse
SELECT is_result_set_caching_on
FROM   sys.databases
WHERE  name = {database_name}

-- Return all query requests with cache hits
-- Run on target data warehouse
SELECT *
FROM   sys.dm_pdw_request_steps
WHERE  command like '%DWResultCacheDb%'
       AND step_index = 0
```

Result-set caching flow

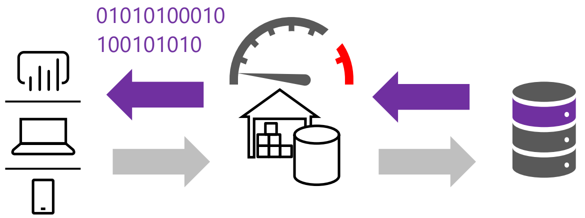


1 Client sends query to DW

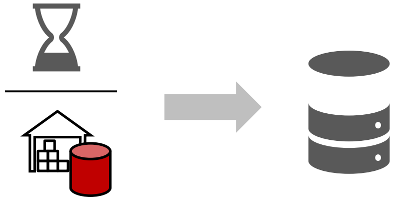
2 Query is processed using DW compute nodes which pull data from remote storage, process query and output back to client app



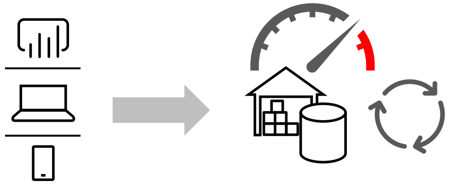
3 Query results are cached in remote storage so subsequent requests can be served immediately



4 Subsequent executions for the same query bypass compute nodes and can be fetched instantly from persistent cache in remote storage



5 Remote storage cache is evicted regularly based on time, cache usage, and any modifications to underlying table data.



6 Cache will need to be regenerated if query results have been evicted from cache

Indexed (materialized) views

Overview

Indexed views cache the schema and data for a view in DW remote storage. They are useful for improving the performance of 'SELECT' statement queries that include aggregations

Indexed views are automatically updated when data in underlying tables are changed. This is a synchronous operation that occurs as soon as the data is changed.

The auto caching functionality allows SQL DW Query Optimizer to consider using indexed view even if the view is not referenced in the query.

Supported aggregations: MAX, MIN, AVG, COUNT, COUNT_BIG, SUM, VAR, STDEV

```
-- Create indexed view
CREATE INDEXED VIEW Sales.vw_Orders
WITH
(
   DISTRIBUTION = ROUND_ROBIN |
   HASH(ProductID)
)
AS
    SELECT SUM(UnitPrice*OrderQty) AS Revenue,
             OrderDate,
            ProductID,
            COUNT_BIG(*) AS OrderCount
    FROM    Sales.SalesOrderDetail
    GROUP   BY OrderDate, ProductID;
GO

-- Disable index view and put it in suspended mode
ALTER INDEX ALL ON Sales.vw_Orders DISABLE;

-- Re-enable index view by rebuilding it
ALTER INDEX ALL ON Sales.vw_Orders REBUILD;
```

Indexed (materialized) views - example

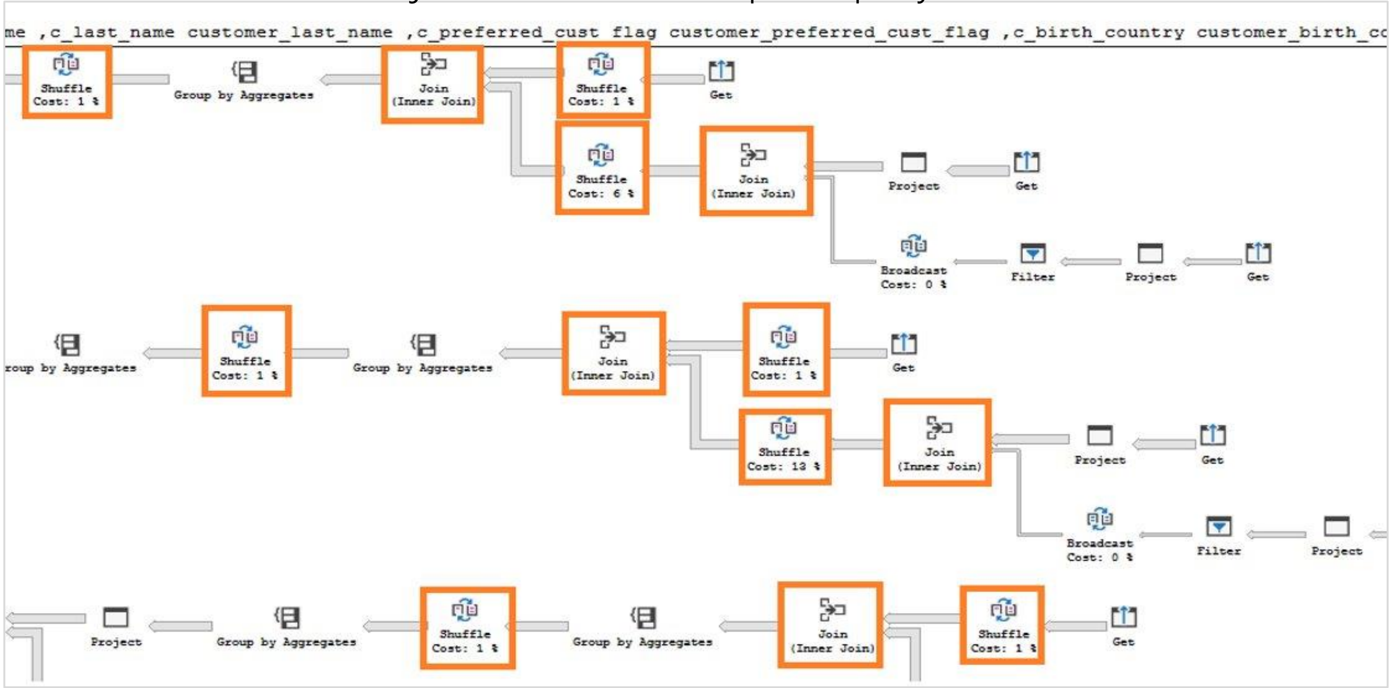
In this example, a query to get the year total sales per customer is shown to have a lot of data shuffles and joins that contribute to slow performance:

No relevant indexed views created on the data warehouse

```
-- Get year total sales per customer
(WITH year_total AS
     SELECT customer_id,
            first_name,
            last_name,
            birth_country,
            login,
            email_address,
            d_year,
            SUM(ISNULL(list_price – wholesale_cost –
            discount_amt + sales_price, 0)/2)year_total
     FROM   customer cust
     JOIN   catalog_sales sales ON cust.sk = sales.sk
     JOIN   date_dim ON sales.sold_date = date_dim.date
     GROUP  BY customer_id, first_name,
           last_name,birth_country,
           login,email_address ,d_year
)
SELECT TOP 100 …
FROM   year_total …
WHERE  …
ORDER BY …
```

Execution time: 103 seconds

Lots of data shuffles and joins needed to complete query



Indexed (materialized) views - example

Now, we add an indexed view to the data warehouse to increase the performance of the previous query. This view can be leveraged by the query even though it is not directly referenced.

Original query – get year total sales per customer

```
-- Get year total sales per customer
(WITH year_total AS
     SELECT customer_id,
            first_name,
            last_name,
            birth_country,
            login,
            email_address,
            d_year,
            SUM(ISNULL(list_price – wholesale_cost –
            discount_amt + sales_price, 0)/2)year_total
     FROM   customer cust
     JOIN   catalog_sales sales ON cust.sk = sales.sk
     JOIN   date_dim ON sales.sold_date = date_dim.date
     GROUP  BY customer_id, first_name,
           last_name,birth_country,
           login,email_address ,d_year
)
SELECT TOP 100 …
FROM   year_total …
WHERE  …
ORDER BY …
```

Create indexed view with hash distribution on customer_id column

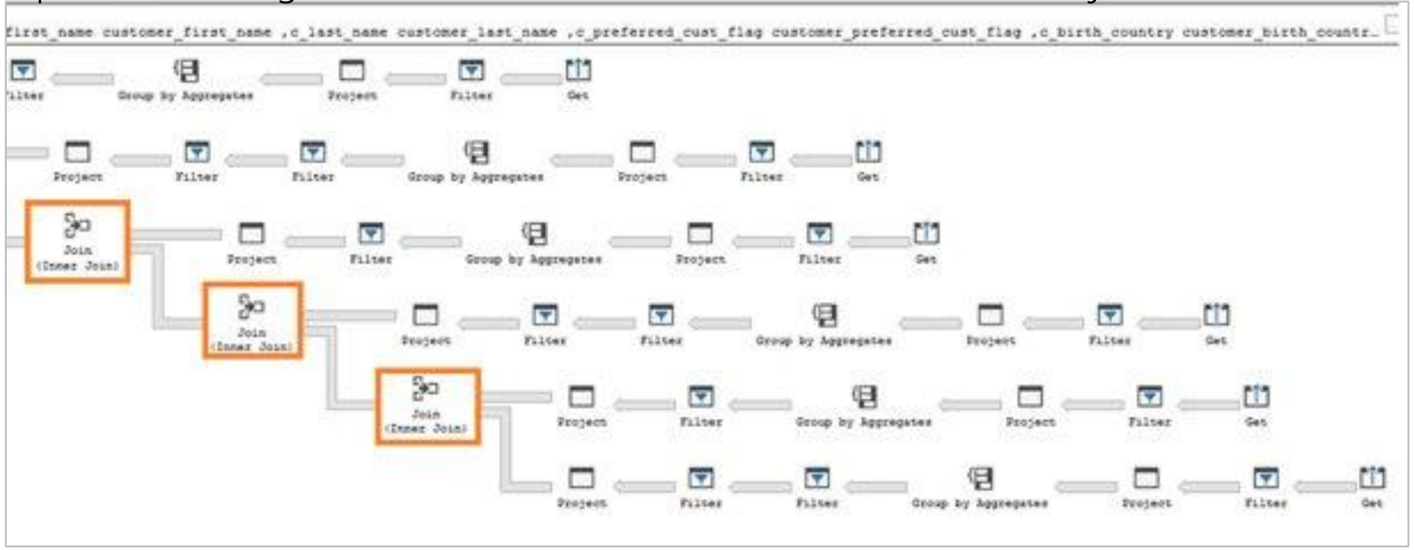
```
-- Create indexed view for query
CREATE INDEXED VIEW nbViewCS WITH (DISTRIBUTION=HASH(customer_id)) AS
SELECT customer_id,
       first_name,
       last_name,
       birth_country,
       login,
       email_address,
       d_year,
       SUM(ISNULL(list_price – wholesale_cost – discount_amt +
       sales_price, 0)/2) AS year_total
FROM   customer cust
JOIN   catalog_sales sales ON cust.sk = sales.sk
JOIN   date_dim ON sales.sold_date = date_dim.date
GROUP  BY customer_id, first_name,
       last_name,birth_country,
       login, email_address, d_year
```

Indexed (materialized) views - example

The SQL Data Warehouse query optimizer automatically leverages the indexed view to speed up the same query. Notice that the query does not need to reference the view directly

Execution time: 6 seconds

Optimizer leverages materialized view to reduce data shuffles and joins needed



Original query – no changes have been made to query

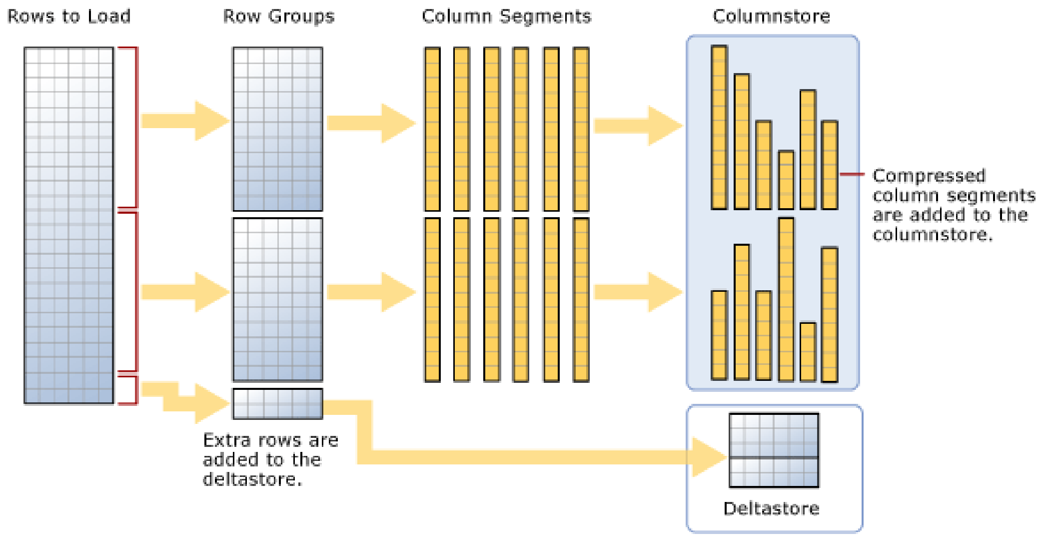
```
-- Get year total sales per customer
(WITH year_total AS
     SELECT customer_id,
            first_name,
            last_name,
            birth_country,
            login,
            email_address,
            d_year,
            SUM(ISNULL(list_price – wholesale_cost –
            discount_amt + sales_price, 0)/2)year_total
     FROM   customer cust
     JOIN   catalog_sales sales ON cust.sk = sales.sk
     JOIN   date_dim ON sales.sold_date = date_dim.date
     GROUP  BY customer_id, first_name,
           last_name,birth_country,
           login,email_address ,d_year
)
SELECT TOP 100 …
FROM   year_total …
WHERE  …
ORDER BY …
```

Ordered Columnstore Segments

Overview

Queries against tables with ordered columnstore segments can take advantage of improved segment elimination to drastically reduce the time needed to service a query.

Columnstore Segments are automatically updated as data is inserted, updated, or deleted in data warehouse tables.




```sql
-- Create Table with Ordered Columnstore Index
CREATE TABLE sortedOrderTable
(
    OrderId   INT NOT NULL,
    Date      DATE NOT NULL,
    Name      VARCHAR(2),
    Country   VARCHAR(2)
)
WITH
(
  CLUSTERED COLUMNSTORE INDEX ORDER (OrderId)
)
-- Create Clustered Columnstore Index on existing table
CREATE CLUSTERED COLUMNSTORE INDEX cciOrderId
ON dbo.OrderTable ORDER (OrderId)

-- Insert data into table with ordered columnstore
index
INSERT INTO sortedOrderTable
VALUES (1, '01-01-2019','Dave’, 'UK')
```

JSON data support – insert JSON data

Overview

The JSON format enables representation of complex or hierarchical data structures in tables.

JSON data is stored using standard NVARCHAR table columns.

```
-- Create Table with column for JSON string
CREATE TABLE CustomerOrders
(
  CustomerId   BIGINT NOT NULL,
  Country      NVARCHAR(150) NOT NULL,
  OrderDetails NVARCHAR(3000) NOT NULL -- NVARCHAR column for JSON
) WITH (DISTRIBUTION = ROUND_ROBIN)

-- Populate table with semi-structured data
INSERT INTO CustomerOrders
VALUES
( 101, -- CustomerId
  'Bahrain', -- Country
  N'[{ StoreId": "AW73565",
       "Order": { "Number":"SO43659",
                  "Date":"2011-05-31T00:00:00"
               },
       "Item":  { "Price":2024.40, "Quantity":1 }
     }]’ -- OrderDetails
)
```

JSON data support – read JSON data

Overview

Read JSON data stored in a string column with the following:

- ISJSON – verify if text is valid JSON
- JSON_VALUE – extract a scalar value from a JSON string
- JSON_QUERY – extract a JSON object or array from a JSON string

```sql
-- Return all rows with valid JSON data
SELECT CustomerId, OrderDetails
FROM   CustomerOrders
WHERE  ISJSON(OrderDetails) > 0;
```

CustomerId	OrderDetails
101	N'[{ StoreId": "AW73565", "Order": { "Number":"SO43659", "Date":"2011-05-31T00:00:00" }, "Item": { "Price":2024.40, "Quantity":1 }}]'

```sql
-- Extract values from JSON string
SELECT CustomerId,
       Country,
       JSON_VALUE(OrderDetails,'$.StoreId') AS StoreId,
       JSON_QUERY(OrderDetails,'$.Item') AS ItemDetails
FROM   CustomerOrders;
```

CustomerId	Country	StoreId	ItemDetails
101	Bahrain	AW73565	{ "Price":2024.40, "Quantity":1 }

JSON data support – modify and operate on JSON data

Overview

Use standard table columns and values from JSON text in the same analytical query.

Modify JSON data with the following:

- JSON_MODIFY – modifies a value in a JSON string
- OPENJSON – convert JSON collection to a set of rows and columns

```
-- Modify Item Quantity value
UPDATE CustomerOrders SET OrderDetails =
JSON_MODIFY(OrderDetails, '$.OrderDetails.Item.Quantity',2)
```

OrderDetails
N'[{ StoreId": "AW73565", "Order": { "Number":"SO43659", "Date":"2011-05-31T00:00:00" }, "Item": { "Price":2024.40, "Quantity": 2}}]'

```
-- Convert JSON collection to rows and columns
SELECT CustomerId,
       StoreId,
       OrderDetails.OrderDate,
       OrderDetails.OrderPrice
FROM   CustomerOrders
CROSS APPLY OPENJSON (CustomerOrders.OrderDetails)
WITH ( StoreId       VARCHAR(50)  '$.StoreId',
       OrderNumber   VARCHAR(100) '$.Order.Date',
       OrderDate     DATETIME     '$.Order.Date',
       OrderPrice    DECIMAL      '$.Item.Price',
       OrderQuantity INT          '$.Item.Quantity'
      ) AS OrderDetails
```

CustomerId	StoreId	OrderDate	OrderPrice
101	AW73565	2011-05-31T00:00:00	2024.40

Windowing functions

OVER clause

Defines a window or specified set of rows within a query result set

Computes a value for each row in the window

Aggregate functions

COUNT, MAX, AVG, SUM, APPROX_COUNT_DISTINCT, MIN, STDEV, STDEVP, STRING_AGG, VAR, VARP, GROUPING, GROUPING_ID, COUNT_BIG, CHECKSUM_AGG

Analytical functions

LAG, LEAD, FIRST_VALUE, LAST_VALUE, CUME_DIST, PERCENTILE_CONT, PERCENTILE_DISC, PERCENT_RANK

Ranking functions

RANK, NTILE, DENSE_RANK, ROW_NUMBER

ROWS | RANGE

PRECEDING, UNBOUNDING PRECEDING, CURRENT ROW, BETWEEN, FOLLOWING, UNBOUNDED FOLLOWING

```sql
SELECT
   ROW_NUMBER() OVER(PARTITION BY PostalCode ORDER BY SalesYTD DESC
) AS "Row Number",
   LastName,
   SalesYTD,
   PostalCode
FROM Sales
WHERE SalesYTD <> 0
ORDER BY PostalCode;
```

Row Number	LastName	SalesYTD	PostalCode
1	Mitchell	4251368.5497	98027
2	Blythe	3763178.1787	98027
3	Carson	3189418.3662	98027
4	Reiter	2315185.611	98027
5	Vargas	1453719.4653	98027
6	Ansman-Wolfe	1352577.1325	98027
1	Pak	4116870.2277	98055
2	Varkey Chudukaktil	3121616.3202	98055
3	Saraiva	2604540.7172	98055
4	Ito	2458535.6169	98055
5	Valdez	1827066.7118	98055
6	Mensa-Annan	1576562.1966	98055
7	Campbell	1573012.9383	98055
8	Tsoflias	1421810.9242	98055

Approximate execution

HyperLogLog accuracy

Will return a result with a 2% accuracy of true cardinality on average.

e.g. COUNT (DISTINCT) returns 1,000,000, HyperLogLog will return a value in the range of 999,736 to 1,016,234.

APPROX_COUNT_DISTINCT

Returns the approximate number of unique non-null values in a group.

Use Case: Approximating web usage trend behavior

```
-- Syntax
APPROX_COUNT_DISTINCT ( expression )

-- The approximate number of different order keys by order status from the orders table.
SELECT O_OrderStatus, APPROX_COUNT_DISTINCT(O_OrderKey) AS Approx_Distinct_OrderKey
FROM dbo.Orders
GROUP BY O_OrderStatus
ORDER BY O_OrderStatus;
```

Group by options

Group by with rollup

Creates a group for each combination of column expressions.

Rolls up the results into subtotals and grand totals.

Grouping sets

Combine multiple GROUP BY clauses into one GROUP BY CLAUSE.

Equivalent of UNION ALL of specified groups.

```
-- GROUP BY SETS Example --
SELECT Country,
SUM(Sales) AS TotalSales
FROM Sales
GROUP BY GROUPING SETS ( Country, () );
```

```
-- GROUP BY ROLLUP Example --
SELECT Country,
Region,
SUM(Sales) AS TotalSales
FROM Sales
GROUP BY ROLLUP (Country, Region);
-- Results --
```

Country	Region	TotalSales
Canada	Alberta	100
Canada	British Columbia	500
Canada	NULL	600
United States	Montana	100
United States	NULL	100
NULL	NULL	700

DATABRICKS – STRUCTURED STREAMING

Overview

The Databricks SQL DW connector supports batch and structured streaming support for writing real-time data into Azure SQL Data Warehouse.

It uses Polybase and the Databricks structured streaming API to stream data from Kafka, Kinesis sources directly into SQL DW at a user-configurable rate.

Source: https://docs.azuredatabricks.net/spark/latest/data-sources/azure/sql-data-warehouse.html#streaming-support

```
# Prepare streaming source; this could be Kafka,
Kinesis, or a simple rate stream.
df = spark.readStream \
  .format("rate") \
  .option("rowsPerSecond", "100000") \
  .option("numPartitions", "16") \
  .load()

# Apply some transformations to the data then use
# Structured Streaming API to continuously write the
data to a table in SQL DW.
df.writeStream \
  .format("com.databricks.spark.sqldw") \
  .option("url", <azure-sqldw-jdbc-url>) \
  .option("tempDir",
"wasbs://<containername>@<storageaccount>.blob.core.
windows.net/<directory>") \
  .option("forwardSparkAzureStorageCredentials",
"true") \
  .option("dbTable", <table-name>) \
  .option("checkpointLocation", "/tmp_location") \
  .start()
```

Since Gen2 GA

05/14/18 Automatic Statistics
06/16/18 User Defined Restore Points
06/21/19 Column-level security
07/25/18 Fast Restore
08/02/18 Recommendations for data skew and table statistics
09/24/18 Streaming support in Azure Databricks
09/24/18 User defined maintenance scheduling
09/24/18 Vulnerability assessment
09/24/18 Intelligent Insights
09/24/18 Flexible Restore Points
11/07/18 RLS
11/12/18 Azure Monitor log support
12/08/18 Azure Virtual Network service endpoints
01/10/19 Azure Data Box Disk

Upcoming Features:
Copy Command
Work Load Management