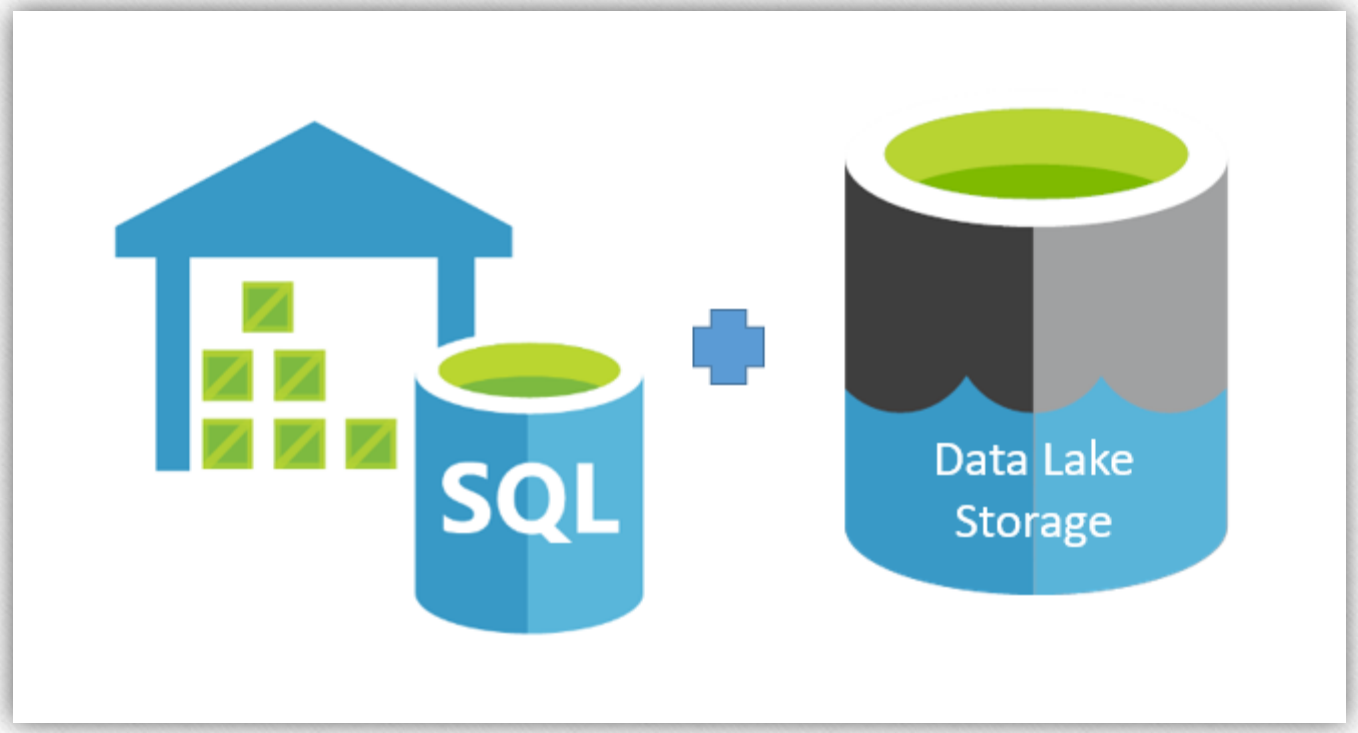
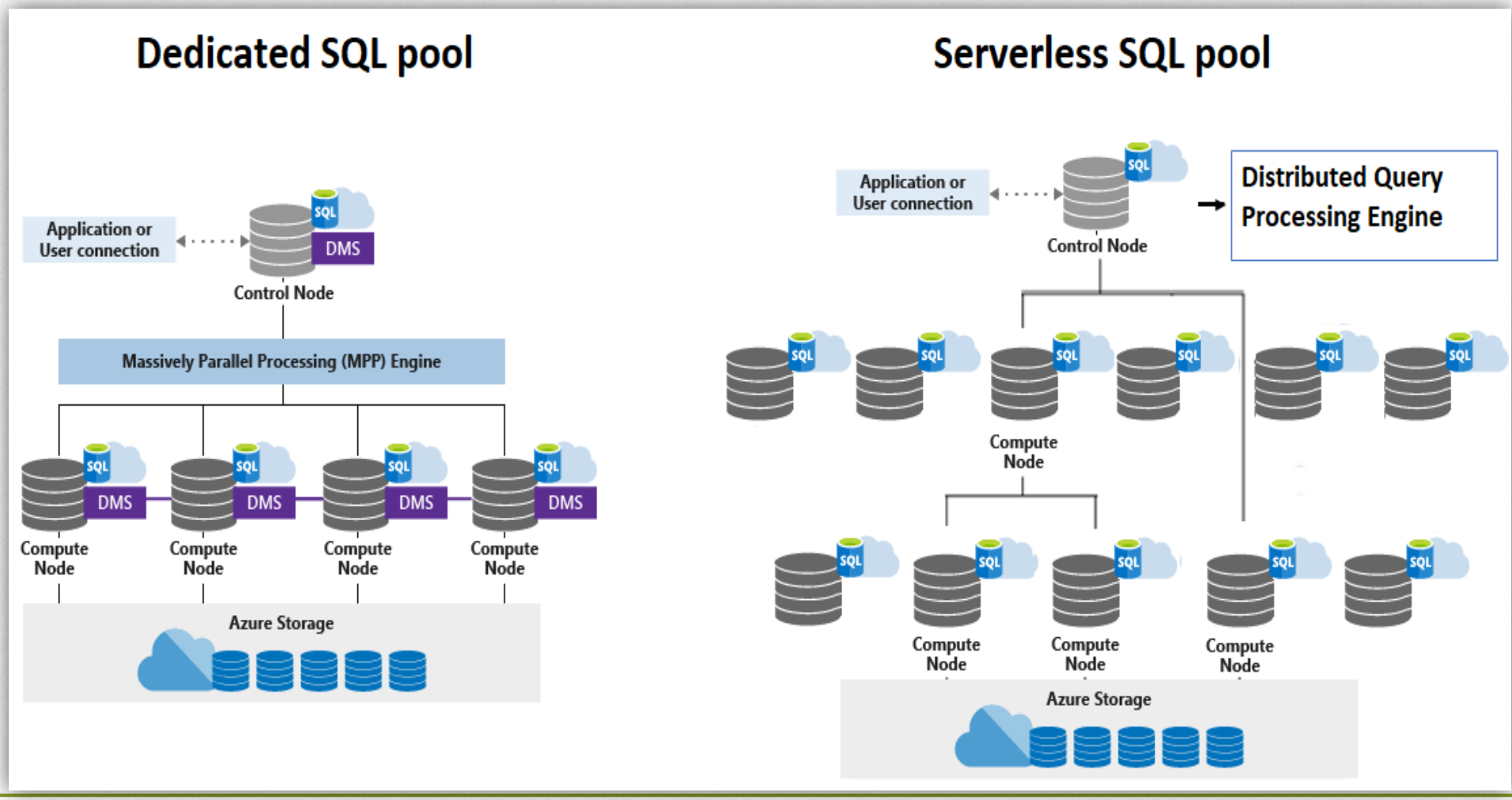
Youtube: TechLake

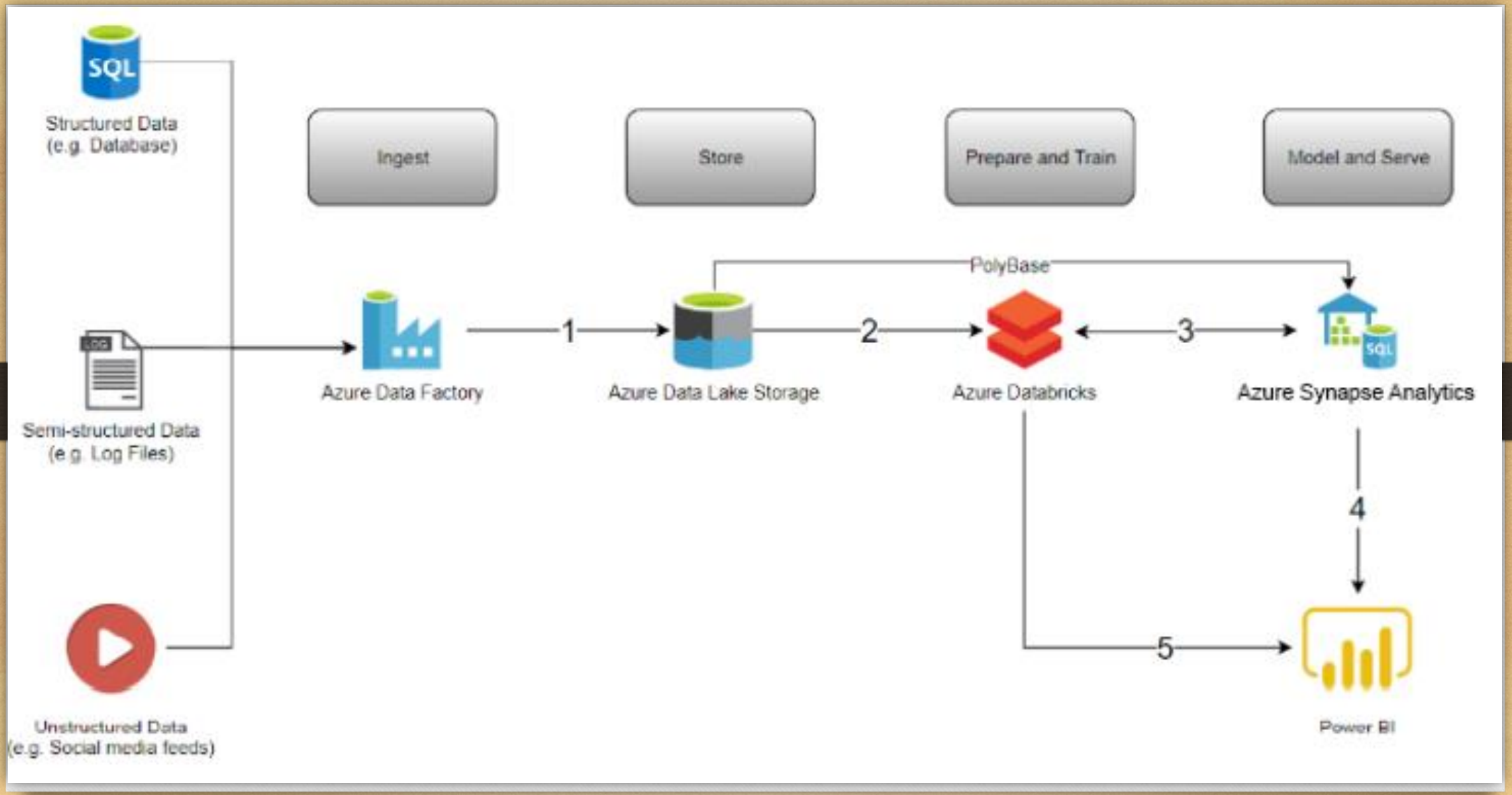
# Synapse SQL DW
# (Dedicated SQL POOL)




Source and Credits : Azure portal and Microsoft standard documentation

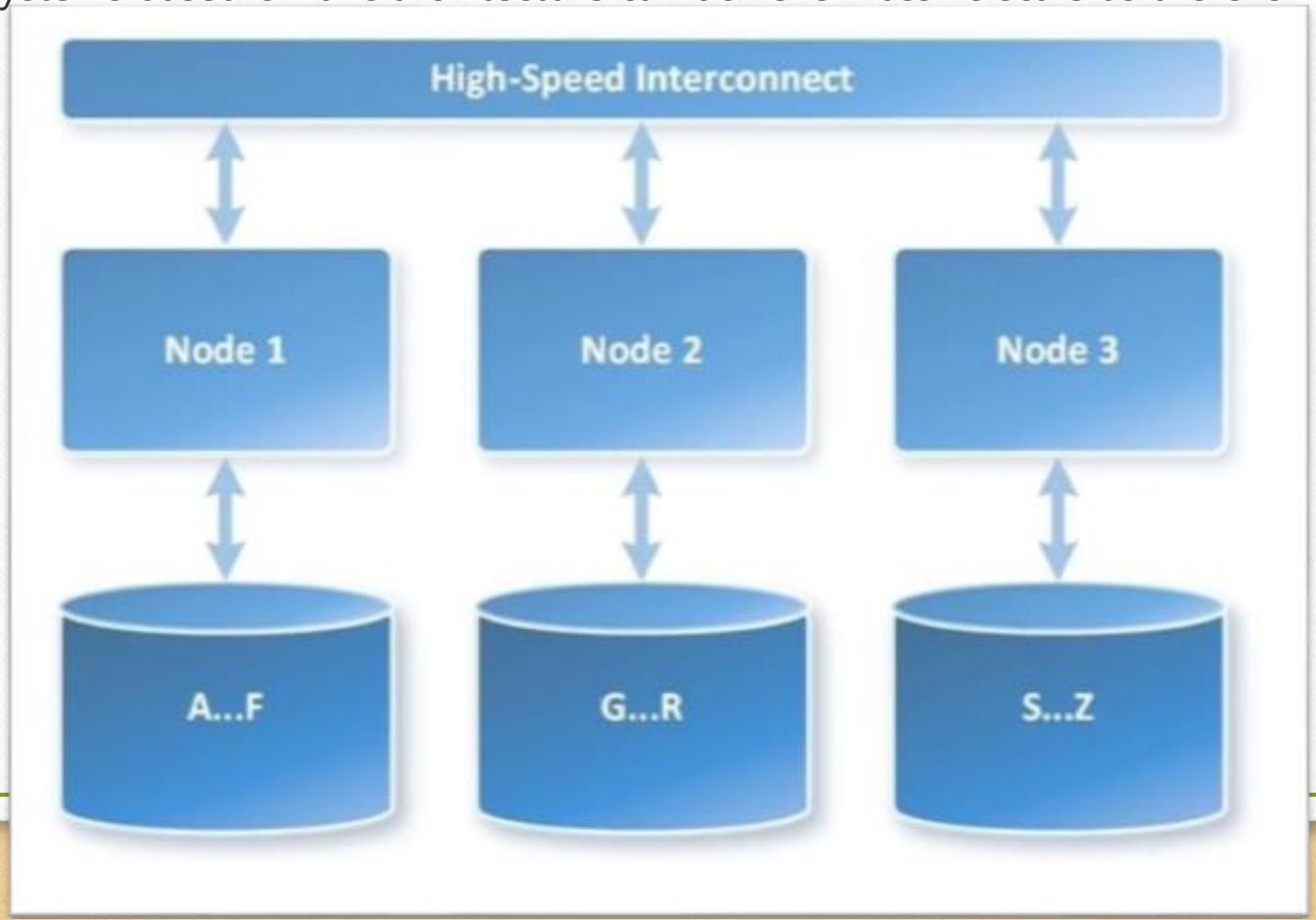
**Azure Synapse (Azure SQL Data Warehouse)** is a **massively parallel processing (MPP)** database system. The data within each synapse instance is spread across 60 underlying databases. These 60 databases are referred to as "distributions". As the data is distributed, there is a need to organize the data in a way that makes querying faster and more efficient.



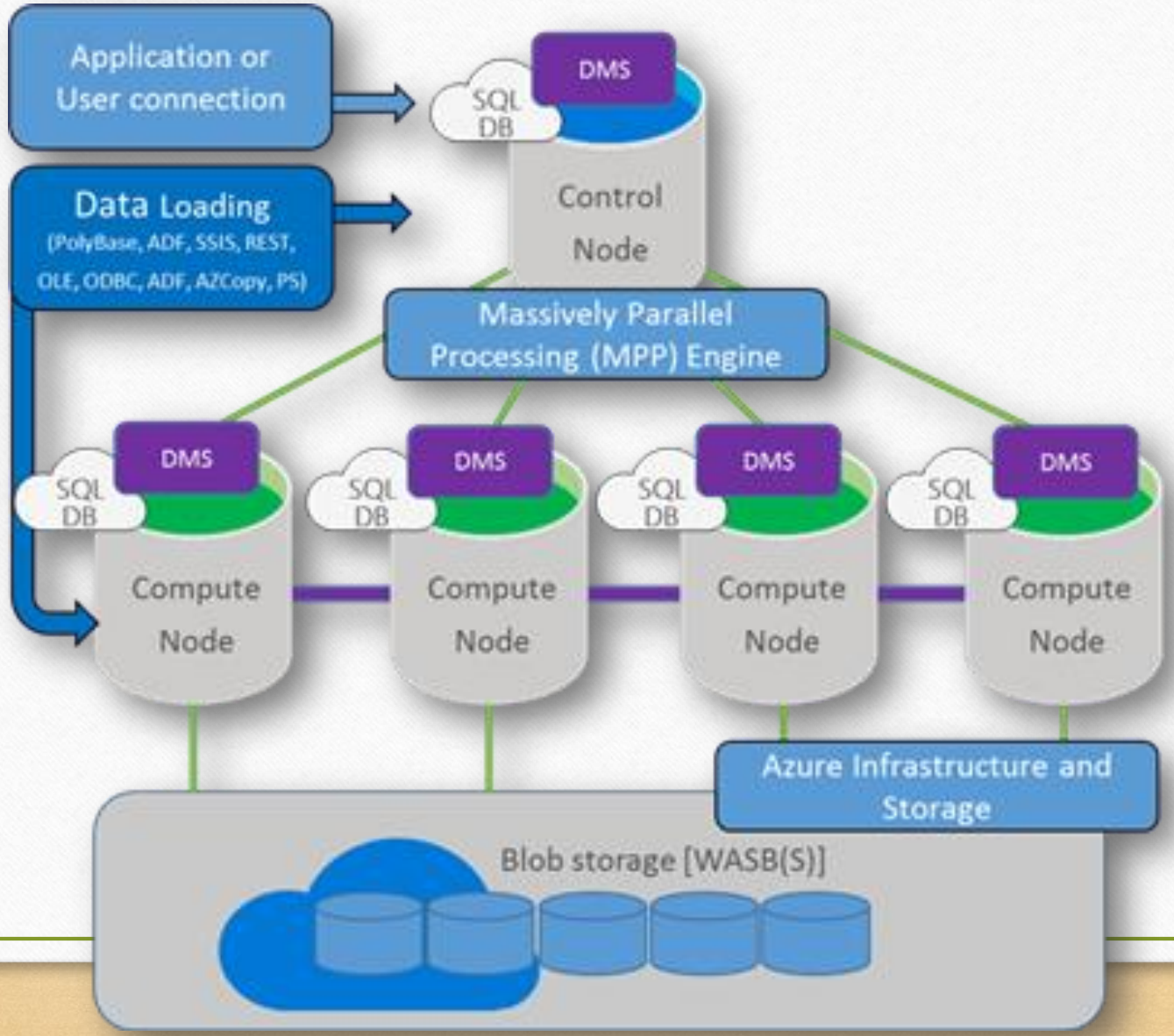

Structured Data
(e.g. Database)
Ingest
Store
Prepare and Train
Model and Serve
PolyBase
Azure Data Factory
1
Azure Data Lake Storage
2
Azure Databricks
3
Azure Synapse Analytics
Semi-structured Data
(e.g. Log Files)
4
5
Unstructured Data
(e.g. Social media feeds)
Power BI

# Massively Parallel Processing (MPP)

**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. MPP can be setup with a shared nothing or shared disk architecture. In a shared nothing architecture, there is no single point of contention across the system and nodes do not share memory or disk storage. Data is horizontally partitioned across nodes, such that each node has a subset of rows from each table in the database. Each node then processes only the rows on its own disks. Systems based on this architecture can achieve massive scale as there is no single bottleneck to slow down the system.



# Synapse SQL DB DW Architecture



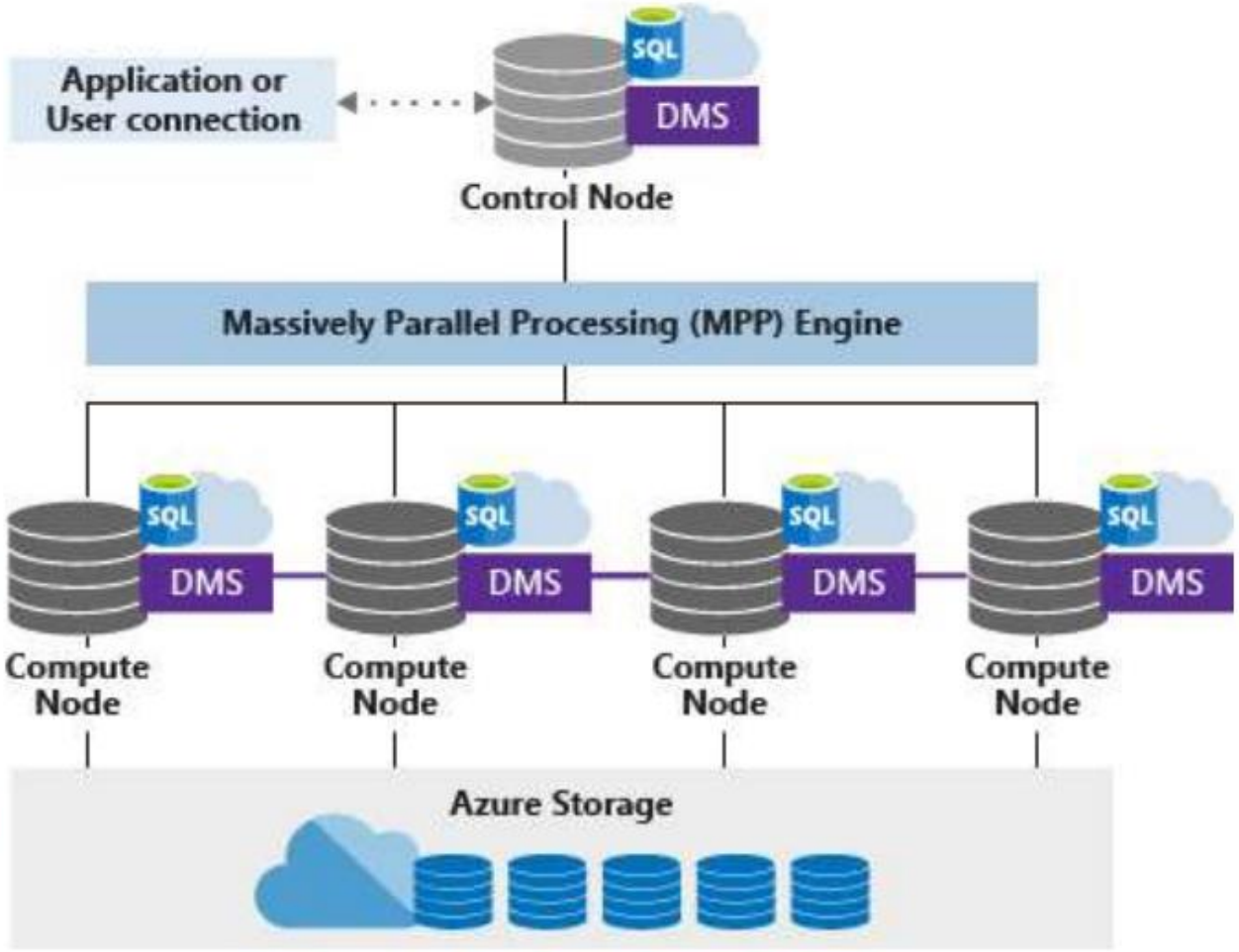
# On-demand MPP databases

- Azure SQL Data Warehouse
- Amazon Redshift
- Google BigQuery
- Snowflake




# Azure Synapse SQL Pool

- Node based
  - Control Node
  - Compute Nodes
- Decoupled Storage
  - Distributed Datafiles
  - Blob Storage
  - Local NVMe caching



# Azure Synapse Analytics Compute

- Control Node:
  - Single Master funnels queries and returns data
  - Governed by AAD for authentication and RBAC
  - Creates query plan and distributes queries
  - Aggregates/combines results and sends back to client
- Compute Nodes:
  - Perform parallel work (1 to 60 nodes).
  - Interact with dedicated databases.
  - Caches data on local NVMe Storage (gen. 2)

# Azure Synapse Analytics Storage

- Independent to the control and compute nodes
- Data located in 60 distributions
- Distributions attached to 1 to 60 compute nodes
- Leverages Azure BLOB behind the scenes
- Data is Geo-replicated
- 3 types of data distributions: Hash, Round-Robin, Replicated

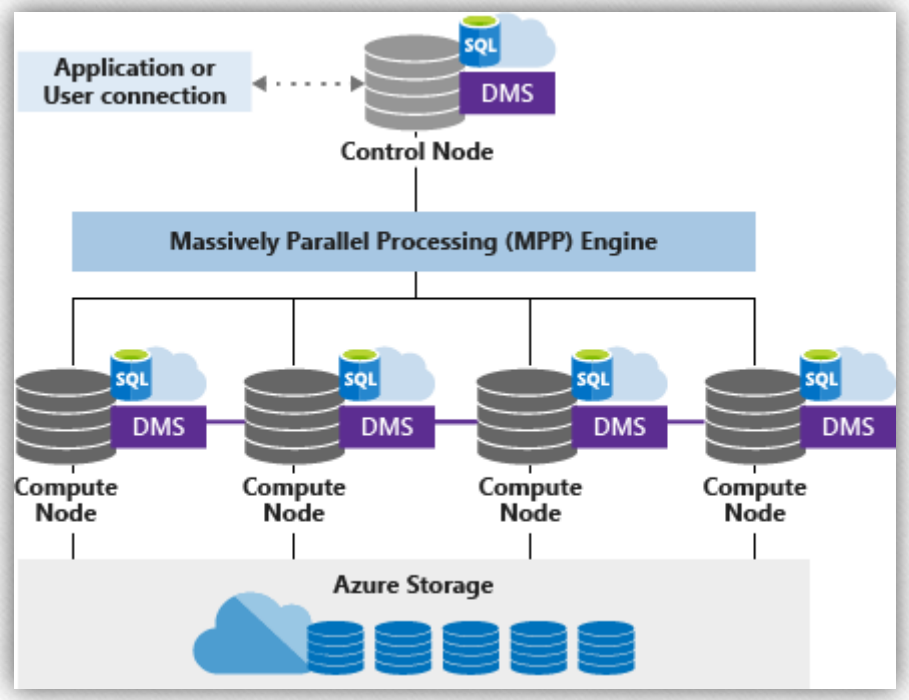
## Data Movement Service

Data Movement Service (DMS) is 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. When data movement is required, DMS ensures the right data gets to the right location.

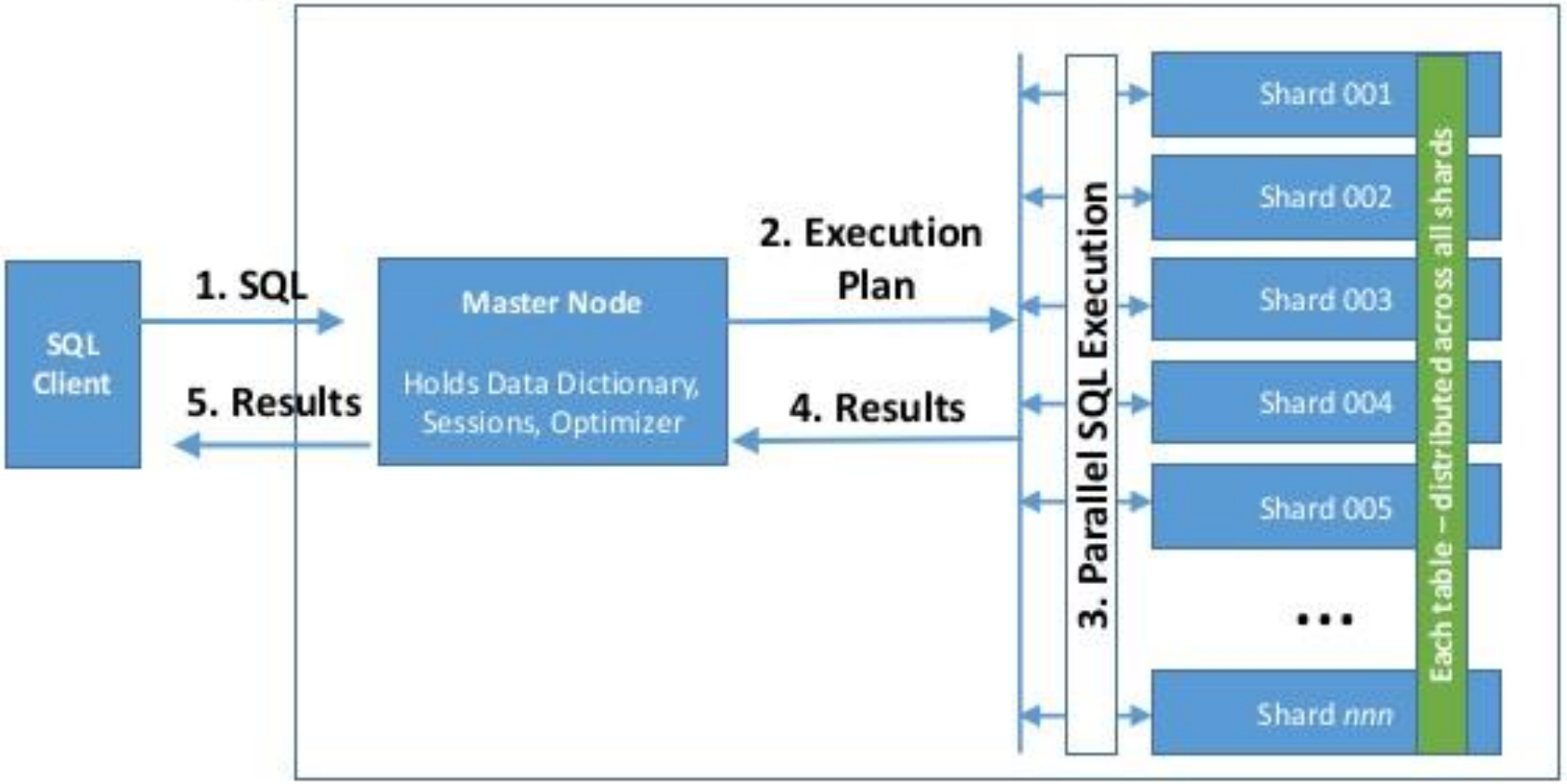
## Distributed Tables

A distributed table is one where row data within the table is distributed across the nodes within the appliance to allow for massive scale. Each row ends up in a one distribution in one compute node

1. Hash
2. Round Robin
3. Replicate



# Sample MPP database architecture



# Azure Synapse Distribution

Distribution is the basic unit for Storage and processing for parallel queries to Distribute your data in multiple Compute node, and when you run a query on Azure synapse it is divided or splitted into 60 smaller queries that run in parallel, to get your results faster, and you can choose Which pattern you need it to distribute your data and this depends on the usage of this data and the table size

## Distributed tables in Azure Synapse

With Azure Synapse we have 3 patterns you can choose between them

1) Hash Distribution
2) Replicate Distribution
3) Round Robin Distribution

# Common table distribution methods

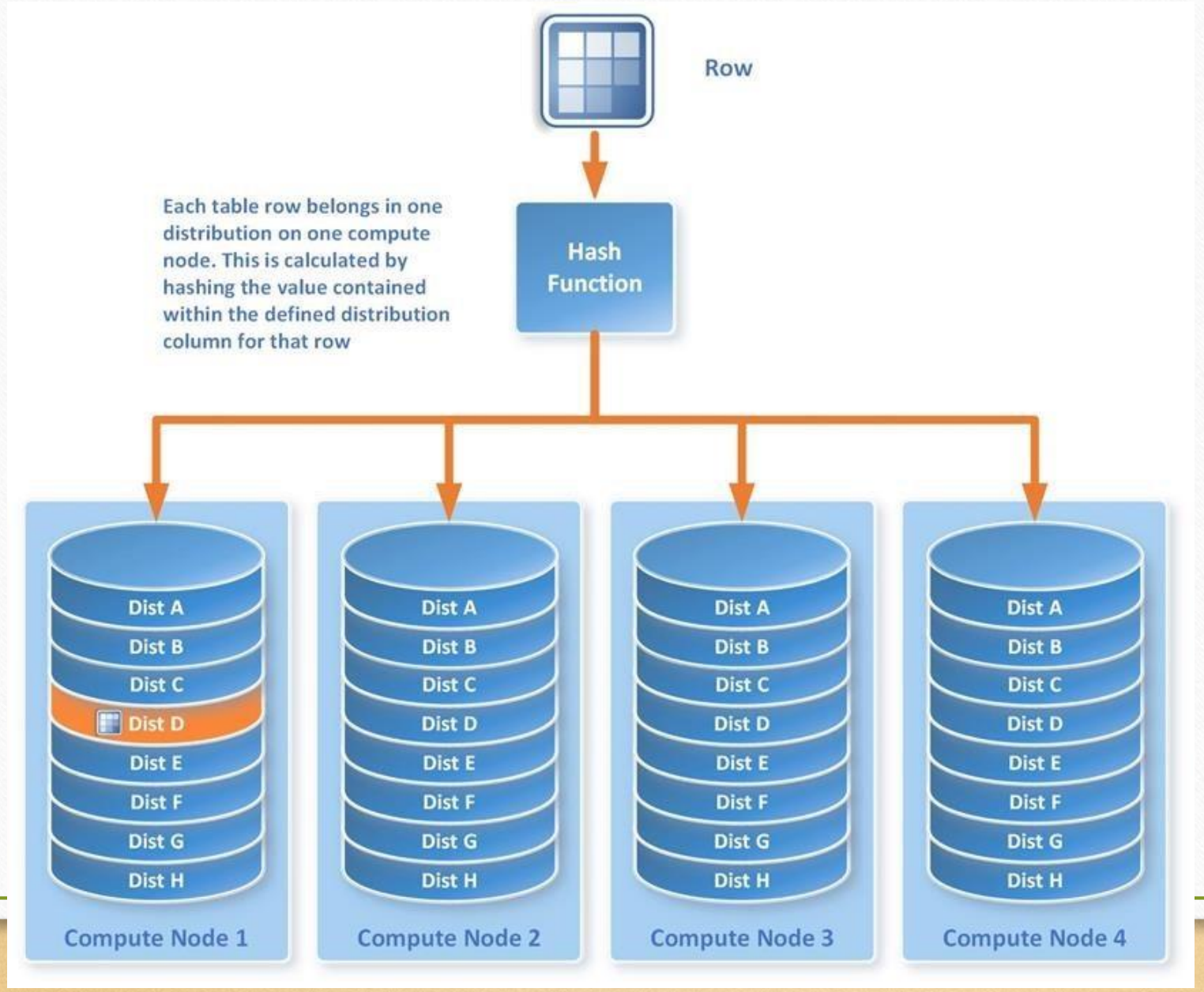
| Table Category | Recommended Distribution Option |
| --- | --- |
| **Fact** | Use hash-distribution with clustered columnstore index. Performance improves because hashing enables the platform to localize certain operations within the node itself during query execution.<br>Operations that benefit:<br>COUNT(DISTINCT( <hashed_key> ))<br>OVER PARTITION BY <hashed_key><br>most JOIN <table_name> ON <hashed_key><br>GROUP BY <hashed_key> |
| **Dimension** | Use replicated for smaller tables. If tables are too large to store on each Compute node, use hash-distributed. |
| **Staging** | Use round-robin for the staging table. The load with CTAS is faster. Once the data is in the staging table, use INSERT...SELECT to move the data to production tables. |

# How to choose best distribution column?

Please make sure to keep these points in consideraion to identify the best distribution column.

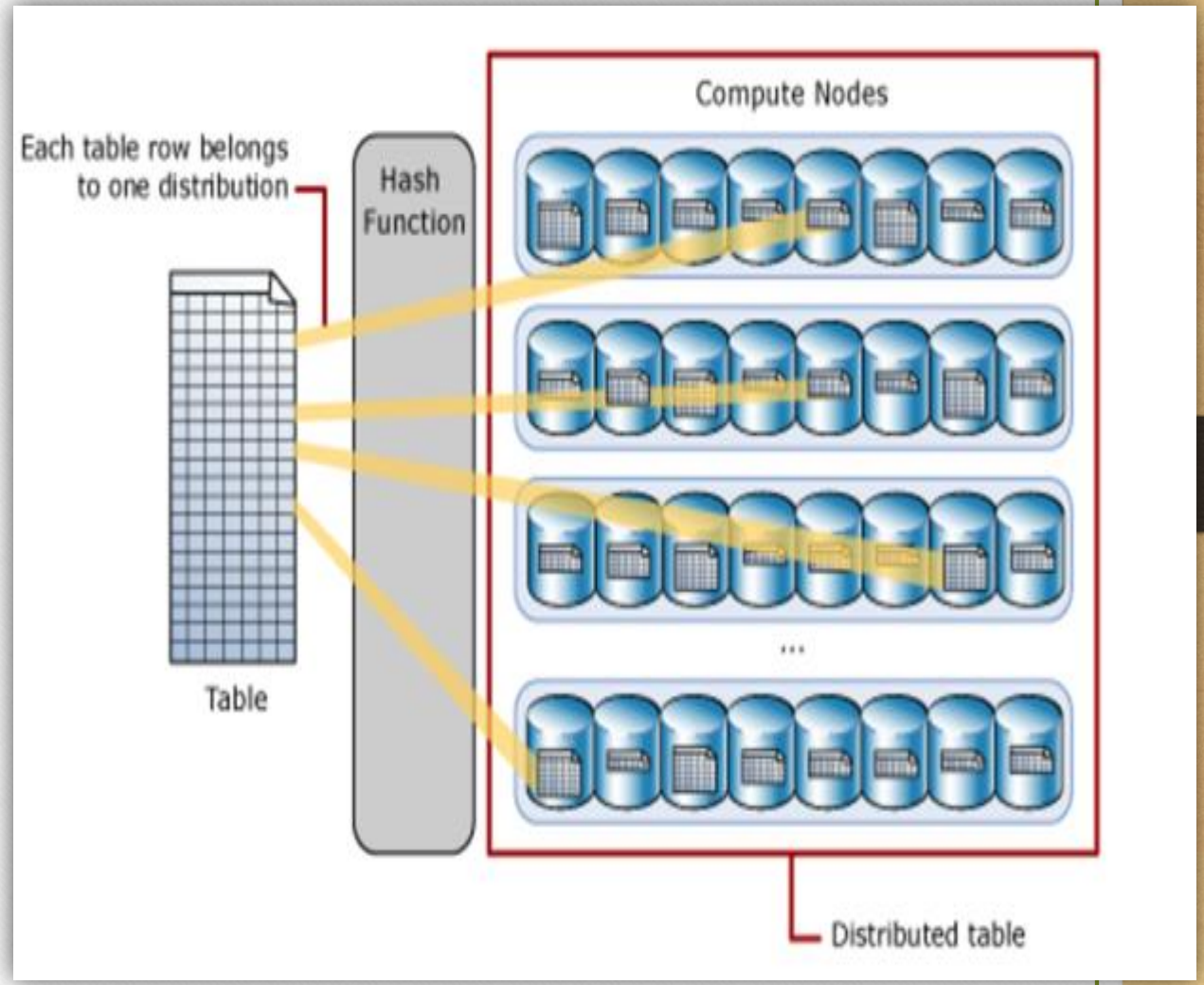
- Choose a column that won't be updated.
- Always choose a column that leads to even distribution (skew) of rows across all the databases.
- Choose an appropriate distribution column that reduces the data movement activity between different distributions.
- Do not choose nullable column because all null columns are hashed in the same way and thus the rows will end up in the same location. If most columns of the table are nulls then it may not be a good candidate for hash distribution.
- Any fact tables that has a default value in a column is also not a good candidate to create a hash distributed table.
- Large fact tables or historical transaction tables are usually stored as hash distributed tables.

# Hash-distributed

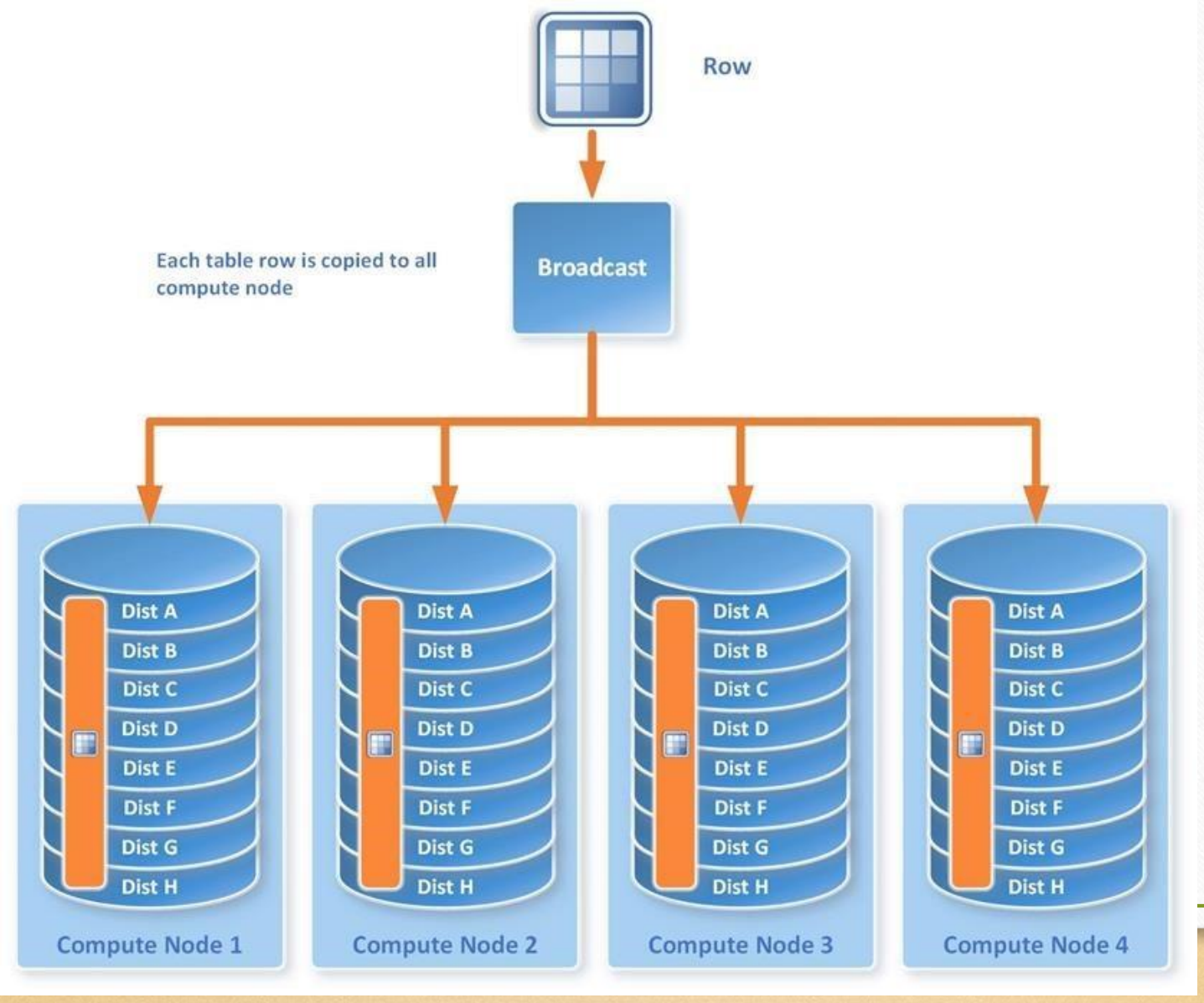


# Hash-distributed

**Hash-Tables** each of these rows are assigned to Specific compute node using deterministic Hash Function and in the table, there is one column defined as distribution column and this deterministic Hash Function used the values in this column to assign each row to a distribution Compute node. So, this means Rows based on Value and with this mechanism, you can reduce your query execution and get better performance for your queries, Hash table is designed for big data table or tables have frequent OLTP transaction (Update, Delete, insert), So with Hash tables, you can get the Highest query performance for joins and aggregations on large tables. Because when you do a query on Hash tables, Hash Function will direct the query to the exact rows in the specific Compute node

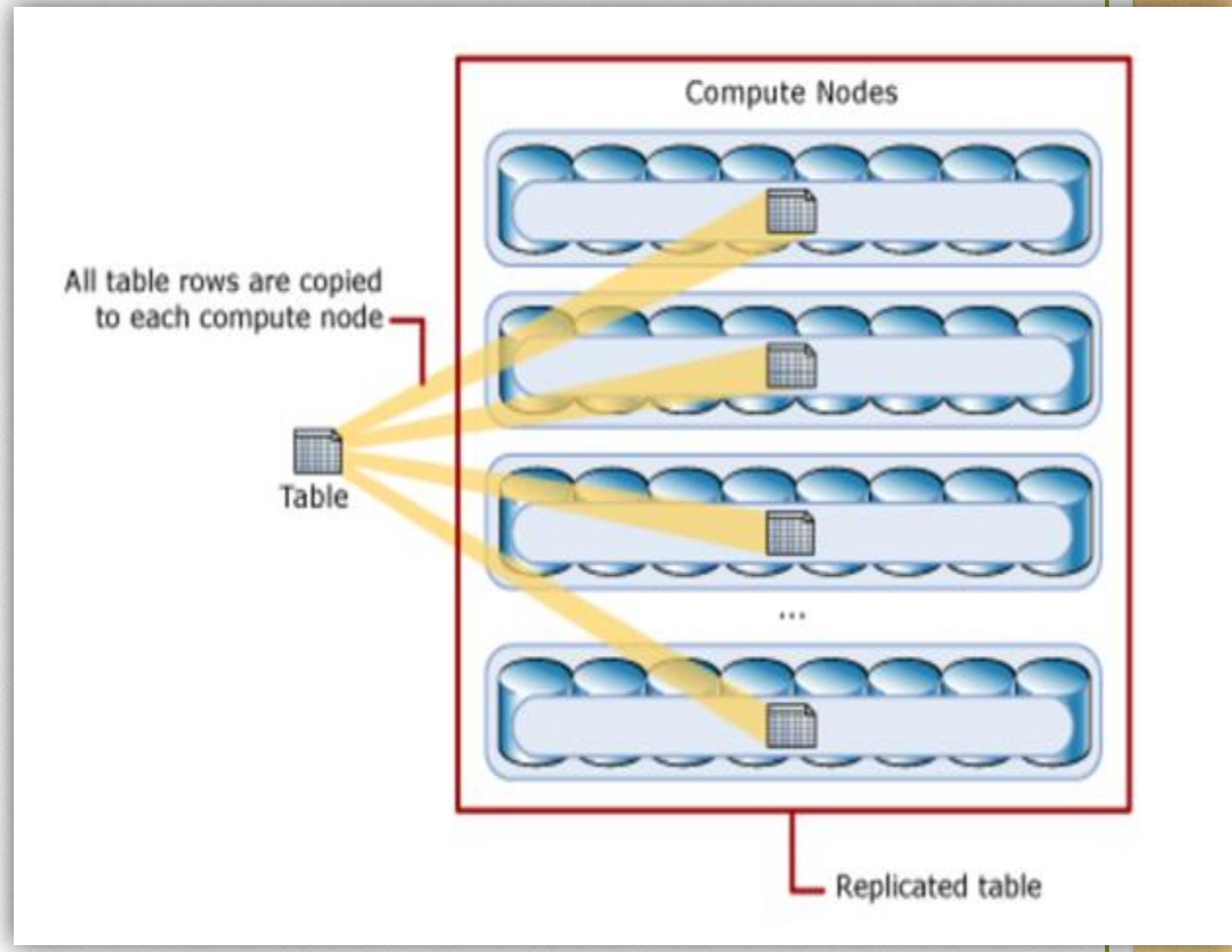


# Replicated Distribution (Broadcast)



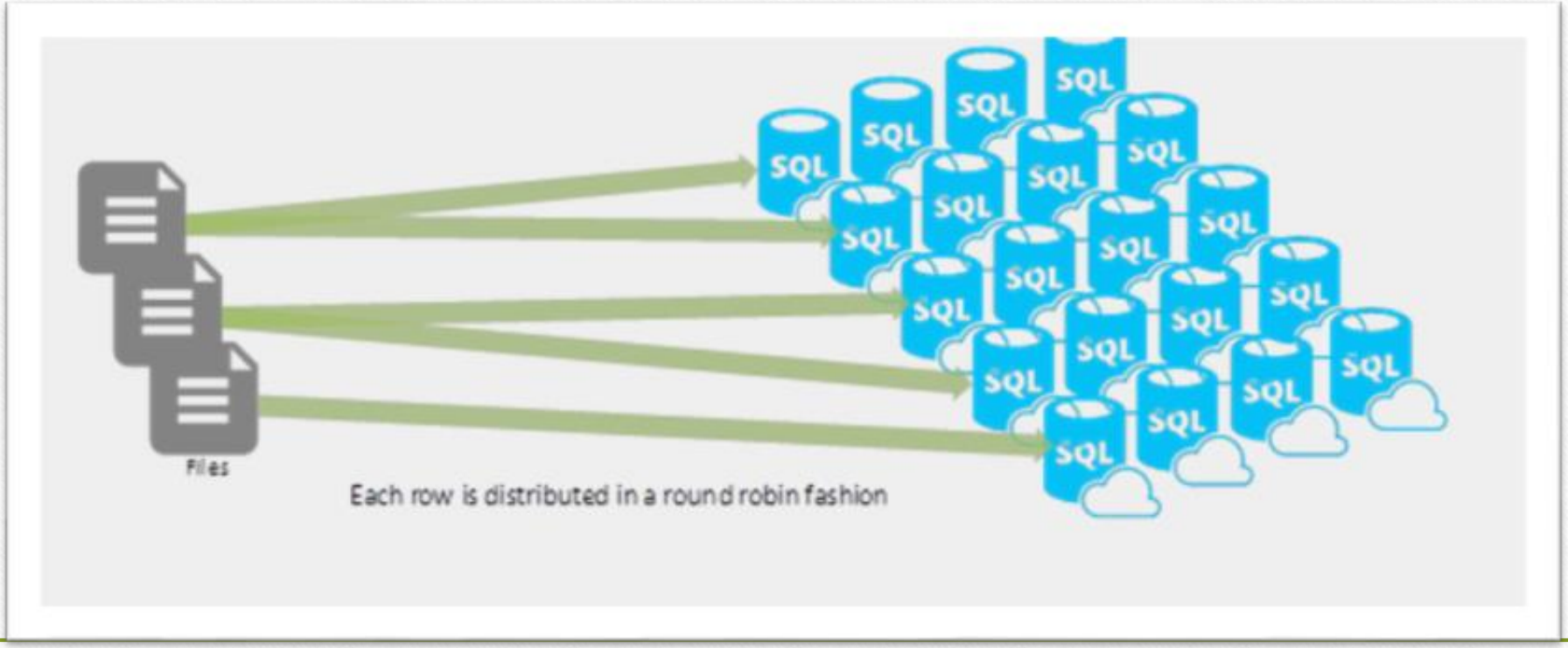
# Replicated Tables

In **Replicated Tables** , all your data will be Fully replicated to all Computed nodes that's why this Replicated tables designed for small tables to get fastest query performance if you design large table as replicated table this mean you required more storage and process to replicated this data. And why it is fast because with a replicated table you don't need to move your data between computed nodes before Join queries because it is already fully copied on all computed nodes that's why it is faster with small tables.



# Round Robin Distribution

When you have staging table used for loading data at this time **Round Robin** tables is the better choice for you to get the best performance, Round Robin Tables somewhat similar Hash-distributed tables but instead of using Hash Function to distributed your rows into computed nodes, it will distribute the table rows equally or evenly across all distributions but this process it's completely random, Also Round tables can be used when you have tables that need a fast load speed and they're a lot of queries running on this table.



**A round-robin table is the most straightforward table to create and delivers fast performance when used as a staging table for loads.**

A round-robin table is the most straightforward table to create and delivers fast performance when used as a staging table for loads. These are some scenarios where you should choose Round robin distribution:

- ✓ When you cannot identify a single key to distribute your data.
- ✓ If your data doesn't frequently join with data from other tables.
- ✓ When there are no obvious keys to join.
- ✓ If the table is being used to hold temporary data.
- ✓ Dimension tables or other lookup tables in a schema can usually be stored as round-robin tables. Usually these tables connects to more than one fact tables and optimizing for one join may not be the best idea. These smaller dimension's tables can leave some distributions empty when has distributed. So use round robin for uniform distribution of such tables.

# Table Distributions

## Round-robin distributed

Distributes table rows evenly across all distributions at random.

## Hash distributed

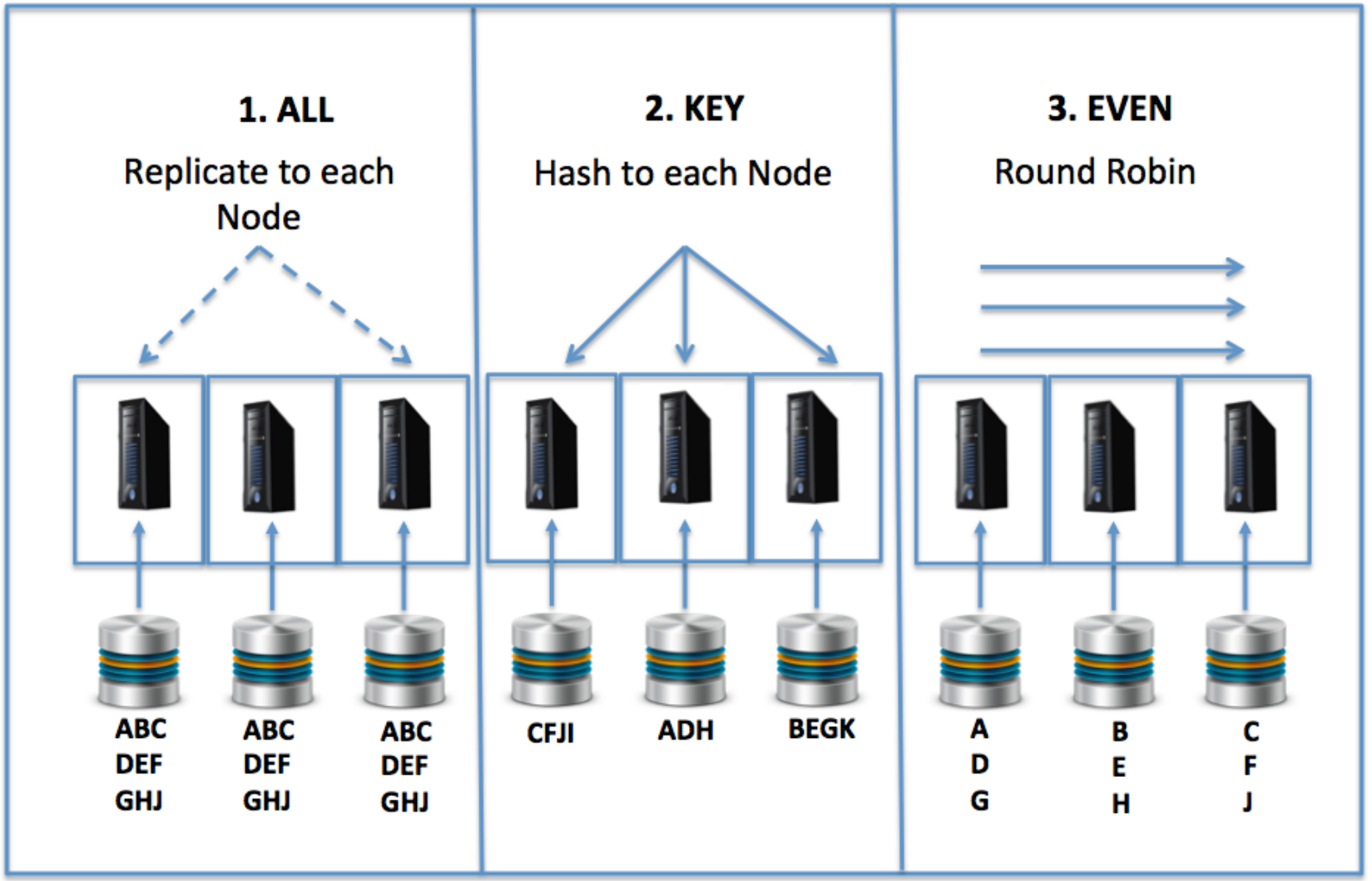
Distributes table rows across the Compute nodes by using a deterministic hash function to assign each row to one distribution.

## Replicated

Full copy of table accessible on each Compute node.

```
CREATE TABLE [dbo].[FactInternetSales]
(
[ProductKey]            int            NOT NULL,
[OrderDateKey]          int            NOT NULL,
[CustomerKey]           int            NOT NULL,
[PromotionKey]          int            NOT NULL,
[SalesOrderNumber]      nvarchar(20)   NOT NULL,
[OrderQuantity]         smallint       NOT NULL,
[UnitPrice]             money          NOT NULL,
[SalesAmount]           money          NOT NULL
)
WITH
(
CLUSTERED COLUMNSTORE INDEX,
DISTRIBUTION = HASH([ProductKey]) |
               ROUND ROBIN |
               REPLICATED
);
```

# Three MPP Data Distribution Styles



| Distribution | Best Fit for… | Do not use when… |
| --- | --- | --- |
| **Replicated** | –Small dimension tables in a star schema with less then 2 GB of storage after the compression (Synapse does 5x compression).<br>-Good for small lookup tables.<br>-Good for dimension tables that are frequently joined with other big tables. | -Many write transactions are on the table (for example insert, delete and updates).<br>-If you change the datawarehouse Units frequently.<br>-You only use 2 -3 columns out of many columns in your tables.<br>-you are indexing a replicated table. |
| **Round Robin (default)** | –Temporary /staging Table.<br>–No obvious joining key candidate is found in the table or If your data doesn't frequently join with data from other tables.<br>–When you cannot identify a single key to distribute your data.<br>–If the table is being used to hold temporary data.<br>–If you are using a staging table for faster loads.<br>–If you are unsure of query patterns and data, you can start with all tables in round-robin distribution. And as you learn the patterns the data can be easily redistributed on a hash key.<br>–Small dimension table. | –Performance is slow due to data movement |
| **Hash** | –Large Fact Tables or historical Transaction tables are good candidates.<br><br>–Large dimension tables. | –The distribution key can not be updated<br>–A nullable column is a bad candidate for any hash distributed table.<br>–Fact tables that has a default value in a column is also not a good candidate to create a hash distributed table. |

# Database Tables

Columnar Storage

Columnar Ordering

Table Partitioning

Hash Distribution

Nonclustered Indexes

## Optimized Storage

Reduce Migration Risk

Less Data Scanned

Smaller Cache Required

Smaller Clusters

Faster Queries

# Index your table

Indexing is helpful for reading tables quickly. There is a unique set of technologies that you can use based on your needs:

| Type | Great fit for... | Watch out if... |
| --- | --- | --- |
| Heap | * Staging/temporary table<br>* Small tables with small lookups | * Any lookup scans the full table |
| Clustered index | * Tables with up to 100 million rows<br>* Large tables (more than 100 million rows) with only 1-2 columns heavily used | * Used on a replicated table<br>* You have complex queries involving multiple join and Group By operations<br>* You make updates on the indexed columns: it takes memory |
| Clustered columnstore index (CCI) (default) | * Large tables (more than 100 million rows) | * Used on a replicated table<br>* You make massive update operations on your table<br>* You overpartition your table: row groups do not span across different distribution nodes and partitions |

# Tables – Indexes

## Clustered Columnstore index (Default Primary)

Highest level of data compression

Best overall query performance

## Clustered index (Primary)

Performant for looking up a single to few rows

## Heap (Primary)

Faster loading and landing temporary data

Best for small lookup tables

## Nonclustered indexes (Secondary)

Enable ordering of multiple columns in a table

Allows multiple nonclustered on a single table

Can be created on any of the above primary indexes

More performant lookup queries

```
-- Create table with index
CREATE TABLE orderTable
(
    OrderId   INT NOT NULL,
    Date      DATE NOT NULL,
    Name      VARCHAR(2),
    Country   VARCHAR(2)
)
WITH
(
    CLUSTERED COLUMNSTORE INDEX |
    HEAP |
    CLUSTERED INDEX (OrderId)
);

-- Add non-clustered index to table
CREATE INDEX NameIndex ON orderTable (Name);
```

# Polybase

**Data ingestion using external data sources**

## Overview

Polybase supports querying files stored in a Hadoop File System (HDFS), Azure Blob storage, or Azure Data Lake Store.

To query files, users create three objects: External data source, external file format, external table.

```
-- Create Azure DataLake Gen2 Storage reference
CREATE EXTERNAL DATA SOURCE AzureStorage with
(
TYPE = HADOOP,
LOCATION='abfss://<container>@<storageaccnt>.blob.core.windows.net',
CREDENTIAL = AzureStorageCredential -- not required if using
managed identity
);
-- Type of format in Hadoop (CSV, RCFILE , ORC, PARQUET).
CREATE EXTERNAL FILE FORMAT TextFileFormat WITH
(
FORMAT_TYPE = DELIMITEDTEXT,
FORMAT_OPTIONS (FIELD_TERMINATOR ='|', USE_TYPE_DEFAULT =
TRUE)
)
-- LOCATION: path to file or directory that contains data
CREATE EXTERNAL TABLE [dbo].[CarSensor_Data]
(
[SensorKey] int NOT NULL,
[Speed] float NOT NULL,
[YearMeasured] int NOT NULL
)
WITH (LOCATION='/Demo/', DATA_SOURCE = AzureStorage,
FILE_FORMAT = TextFileFormat
);
```

# Choosing a Distribution Column

To help choose a distribution column that will result in the best performance, you can follow these guidelines.

**No Updates** - Distribution columns cannot be updated

**Even Distribution of Values** - for best performance, all distributions should have the same number of rows. Queries can only be as fast as the largest distribution. To achieve this, aim for columns that have:

- ✓ **Many unique values** - more uniques, higher chance of evening the distribution
- ✓ **Few or No Nulls**
- ✓ **Not a Date Column** - If all users are filtering on the same date, only one node will be doing all the processing.

**JOIN or Group By Column** - Selecting a Distribution column that is commonly used in a Join or Group by clause reduces the amount of data movement to process a query.

**If no good option** - Create a **Composite Column** using multiple join columns

# Ordered Clustered Columnstore Indexes

## 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.

```
-- 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')
```

# Tables – Partitions

## Overview

Table partitions divide data into smaller groups

In most cases, partitions are created on a date column

Supported on all table types

RANGE RIGHT – Used for time partitions

RANGE LEFT – Used for number partitions

## Benefits

Improves efficiency and performance of loading and querying by limiting the scope to subset of data.

Offers significant query performance enhancements where filtering on the partition key can eliminate unnecessary scans and eliminate IO.

```
CREATE TABLE partitionedOrderTable
(
    OrderId   INT NOT NULL,
    Date      DATE NOT NULL,
    Name      VARCHAR(2),
    Country   VARCHAR(2)
)
WITH
(
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = HASH([OrderId]),
    PARTITION (
    [Date] RANGE RIGHT FOR VALUES (
    '2000-01-01', '2001-01-01', '2002-01-01',
    '2003-01-01', '2004-01-01', '2005-01-01'
    )
  )
);
```

# Tables – Distributions & Partitions

## Logical table structure

| OrderId | Date | Name | Country |
|---|---|---|---|
| 85016 | 11-2-2018 | V | UK |
| 85018 | 11-2-2018 | Q | SP |
| 85216 | 11-2-2018 | Q | DE |
| 85395 | 11-2-2018 | V | NL |
| 82147 | 11-2-2018 | Q | FR |
| 86881 | 11-2-2018 | D | UK |
| 93080 | 11-3-2018 | R | UK |
| 94156 | 11-3-2018 | S | FR |
| 96250 | 11-3-2018 | Q | NL |
| 98799 | 11-3-2018 | R | NL |
| 98015 | 11-3-2018 | T | UK |
| 98310 | 11-3-2018 | D | DE |
| 98979 | 11-3-2018 | Z | DE |
| 98137 | 11-3-2018 | T | FR |
| ... | ... | ... | ... |

## Physical data distribution

( Hash distribution (OrderId), Date partitions )

**Distribution1**
**(OrderId 80,000 – 100,000)**

11-2-2018 partition

| OrderId | Date | Name | Country |
|---|---|---|---|
| 85016 | 11-2-2018 | V | UK |
| 85018 | 11-2-2018 | Q | SP |
| 85216 | 11-2-2018 | Q | DE |
| 85395 | 11-2-2018 | V | NL |
| 82147 | 11-2-2018 | Q | FR |
| 86881 | 11-2-2018 | D | UK |
| ... | ... | ... | ... |

11-3-2018 partition

| OrderId | Date | Name | Country |
|---|---|---|---|
| 93080 | 11-3-2018 | R | UK |
| 94156 | 11-3-2018 | S | FR |
| 96250 | 11-3-2018 | Q | NL |
| 98799 | 11-3-2018 | R | NL |
| 98015 | 11-3-2018 | T | UK |
| 98310 | 11-3-2018 | D | DE |
| 98979 | 11-3-2018 | Z | DE |
| 98137 | 11-3-2018 | T | FR |
| ... | ... | ... | ... |

• • •

x 60 distributions (shards)

- Each shard is partitioned with the same date partitions
- A minimum of 1 million rows per distribution and partition is needed for optimal compression and performance of clustered Columnstore tables

# Resource class types

## Static Resource Classes

Allocate the same amount of memory independent of the current service-level objective (SLO).

Well-suited for fixed data sizes and loading jobs.

## Dynamic Resource Classes

Allocate a variable amount of memory depending on the current SLO.

Well-suited for growing or variable datasets.

All users default to the *smallrc* dynamic resource class.

## Static resource classes:

```
staticrc10 | staticrc20 | staticrc30 |
staticrc40 | staticrc50 | staticrc60 |
staticrc70 | staticrc80
```

---

## Dynamic resource classes:

```
smallrc | mediumrc | largerc | xlargerc
```

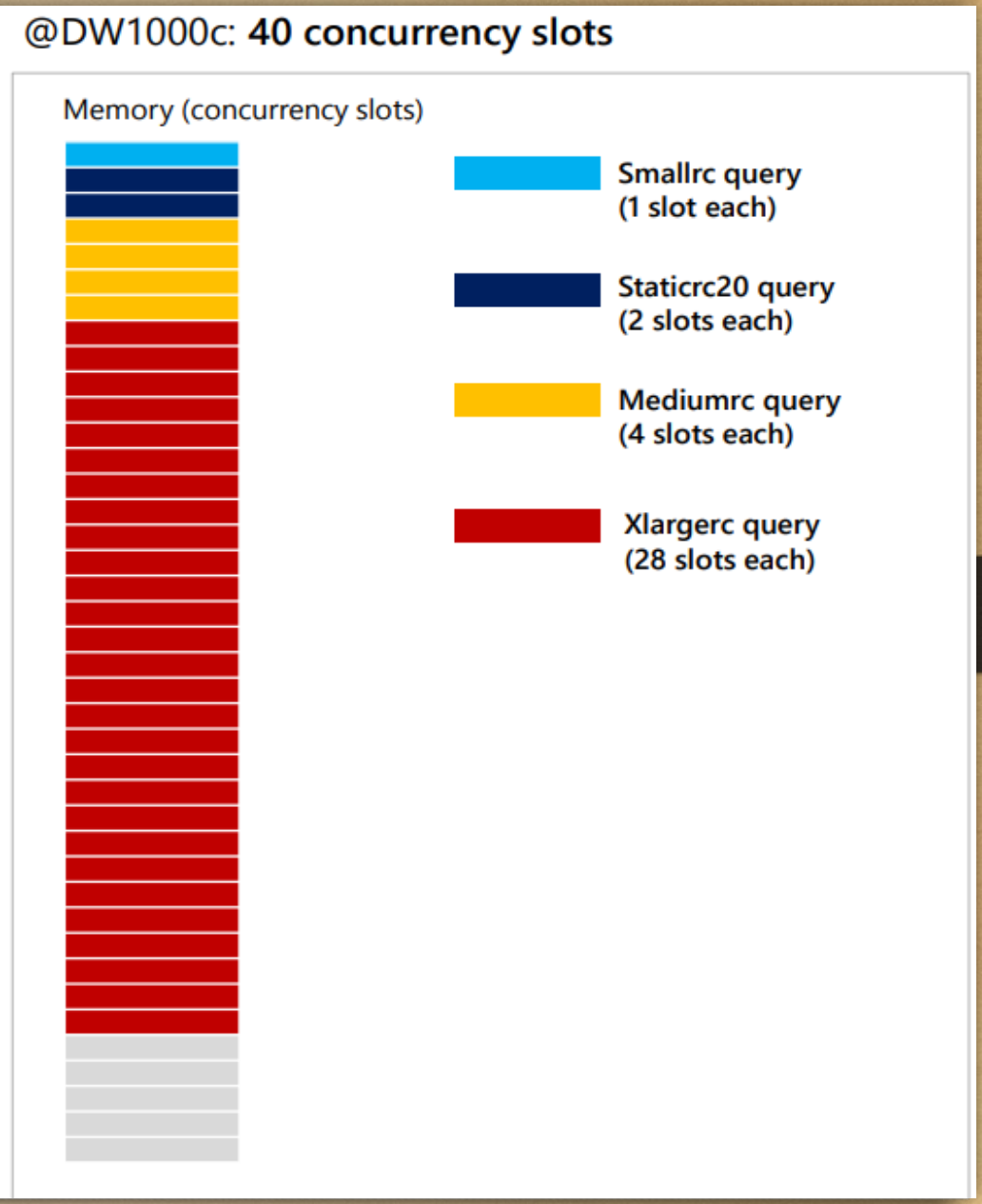
| Resource Class | Percentage Memory | Max. Concurrent Queries |
|---|---|---|
| smallrc | 3% | 32 |
| mediumrc | 10% | 10 |
| largerc | 22% | 4 |
| xlargerc | 70% | 1 |

# Concurrency slots

## Overview

Queries running on a DW compete for access to system resources (CPU, IO, and memory).

To guarantee access to resources, running queries are assigned a chunk of system memory (**a concurrency slot**) for processing the query. The amount given is determined by the resource class of the user executing the query. Higher DW SLOs provide more memory and concurrency slots



# Concurrent query limits

## Overview

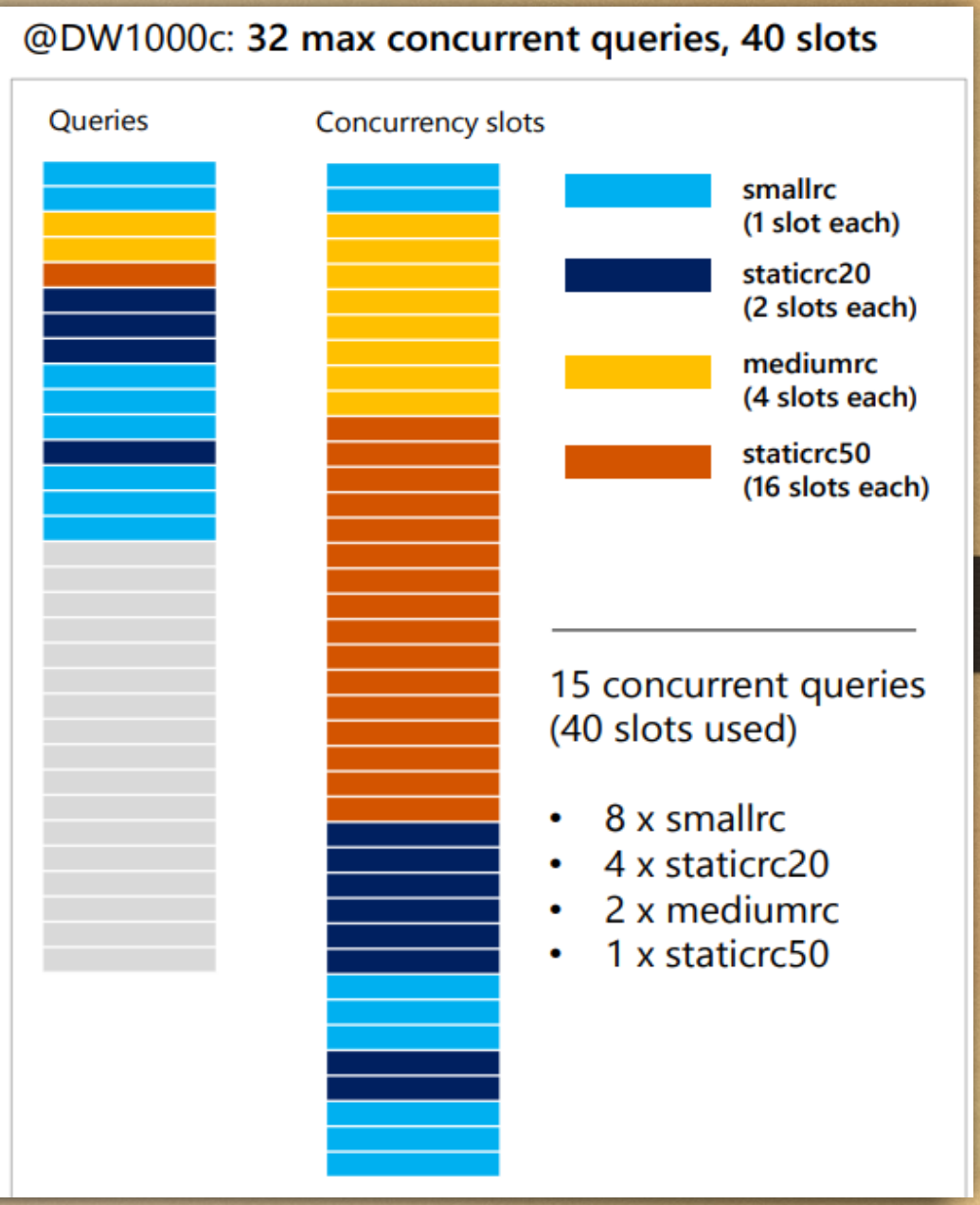
The limit on how many queries can run at the same time is governed by two properties:

- The max. concurrent query count for the DW SLO
- The total available memory (concurrency slots) for the DW SLO

Increase the concurrent query limit by:

- Scaling up to a higher DW SLO (up to 128 concurrent queries)
- Using lower resource classes that use less memory per query

[Concurrency limits based on resource classes]



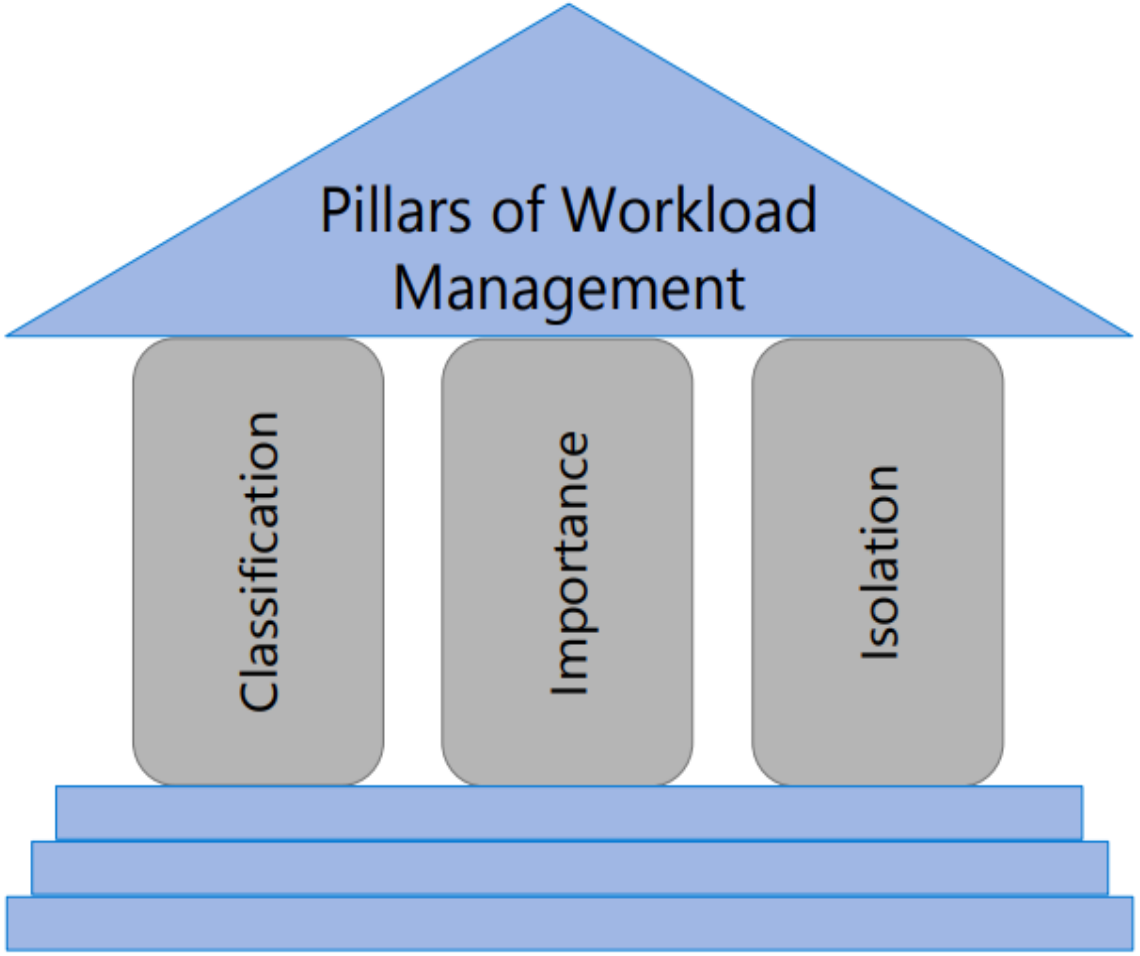
# Workload Management

## Overview

It manages resources, ensures highly efficient resource utilization, and maximizes return on investment (ROI).

The three pillars of workload management are

1. Workload Classification – To assign a request to a workload group and setting importance levels.
2. Workload Importance – To influence the order in which a request gets access to resources.
3. Workload Isolation – To reserve resources for a workload group.



# Workload classification

## Overview

Map queries to allocations of resources via pre-determined rules.

Use with workload importance to effectively share resources across different workload types.

If a query request is not matched to a classifier, it is assigned to the default workload group (smallrc resource class).

## Benefits

Map queries to both Resource Management and Workload Isolation concepts.

Manage groups of users with only a few classifiers.

## Monitoring DMVs

`sys.workload_management_workload_classifiers`
`sys.workload_management_workload_classifier_details`
Query DMVs to view details about all active workload classifiers.

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

*WORKLOAD_GROUP: maps to an existing resource class*
*IMPORTANCE: specifies relative importance of request*
*MEMBERNAME: database user, role, AAD login or AAD group*

# Workload importance

## Overview

Queries past the concurrency limit enter a FiFo queue

By default, queries are released from the queue on a first-in, first-out basis as resources become available

Workload importance allows higher priority queries to receive resources immediately regardless of queue

## Example Video

State analysts have normal importance.

National analyst is assigned high importance.

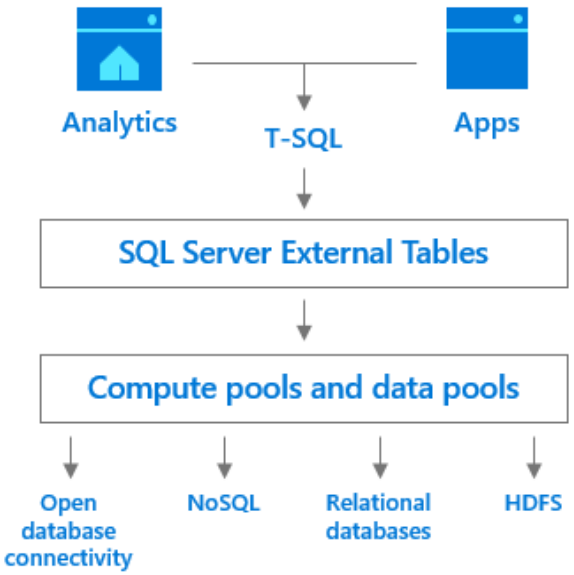
State analyst queries execute in order of arrival

When the national analyst's query arrives, it jumps to the top of the queue

```
CREATE WORKLOAD CLASSIFIER National_Analyst
WITH
(
    [WORKLOAD_GROUP = 'smallrc']
    [IMPORTANCE = HIGH]
    [MEMBERNAME = 'National_Analyst_Login']
```

# SQL Server big data, analytics, and AI

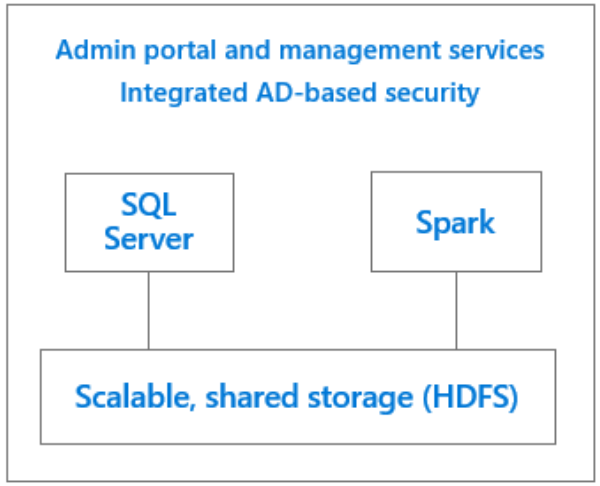
## Data virtualization




Combine data from many sources without moving or replicating it

Scale out compute and caching to boost performance

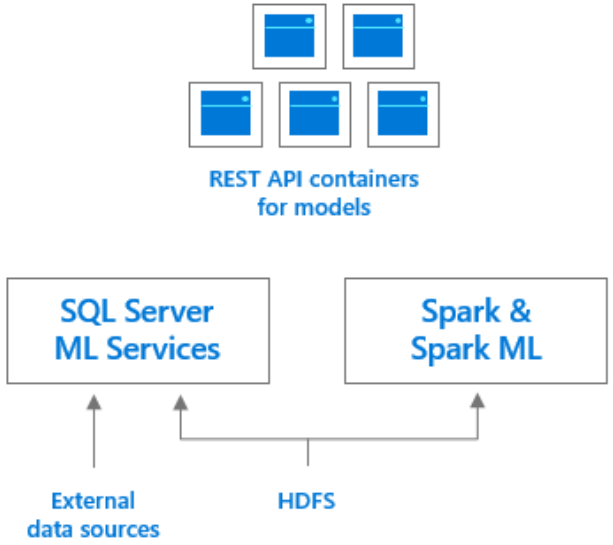
## Managed SQL Server, Spark, and data lake




Store high volume data in a data lake and access it easily using either SQL or Spark

Management services, admin portal, and integrated security make it all easy to manage

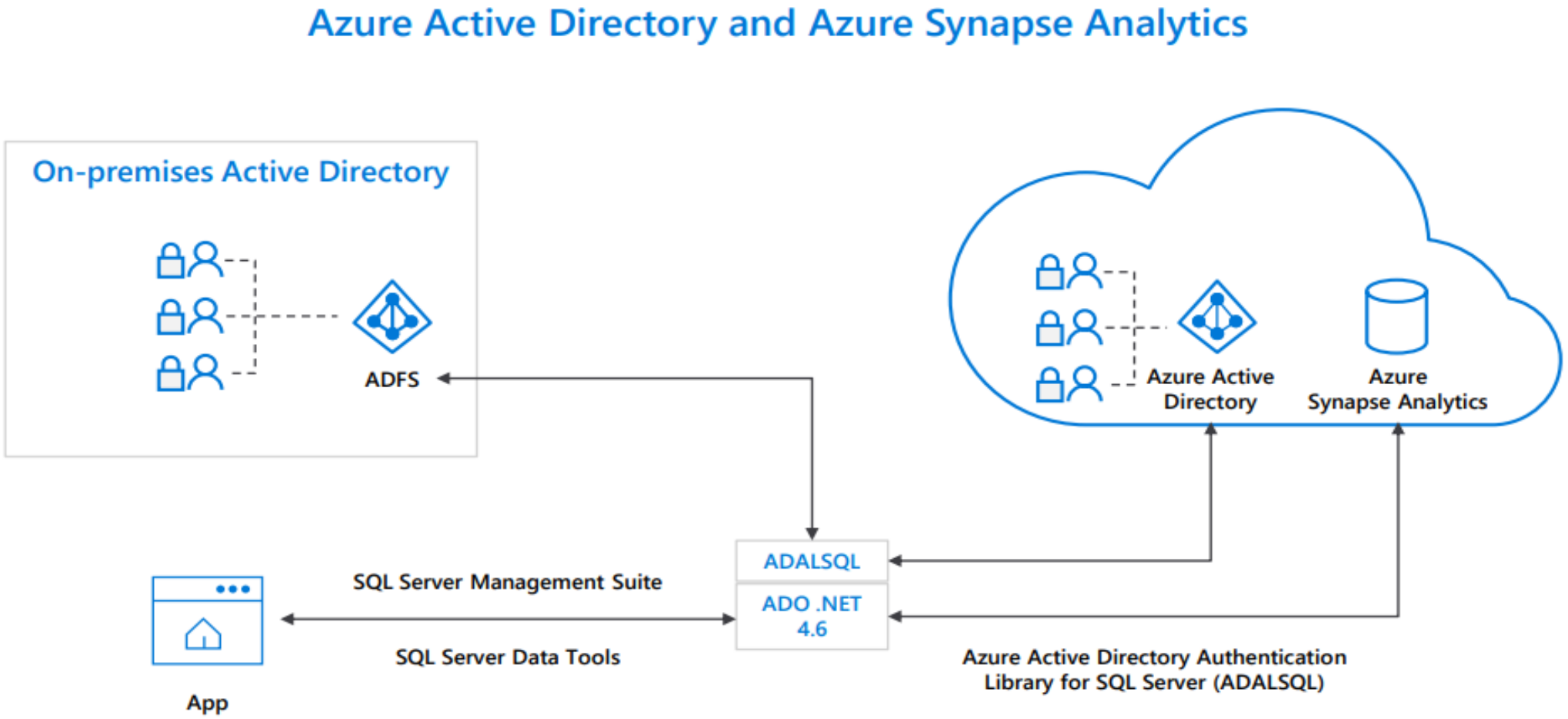
## Complete AI platform




Easily feed integrated data from many sources to your model training

Ingest and prep data and then train, store, and operationalize your models all in one system

# Azure Active Directory trust architecture



# SQL authentication

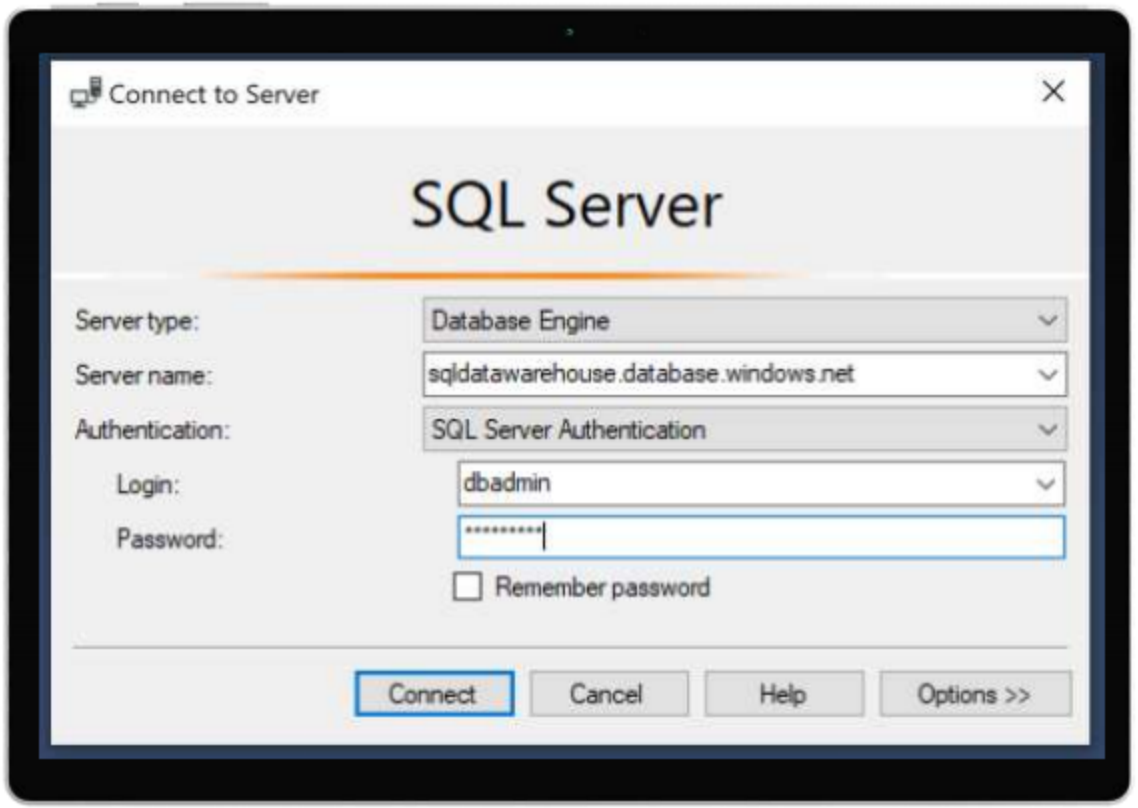
## Overview

This authentication method uses a username and password.

When you created the logical server for your data warehouse, you specified a "server admin" login with a username and password.

Using these credentials, you can authenticate to any database on that server as the database owner.

Furthermore, you can create user logins and roles with familiar SQL Syntax.




```
-- Connect to master database and create a login
CREATE LOGIN ApplicationLogin WITH PASSWORD = 'Str0ng_password';
CREATE USER ApplicationUser FOR LOGIN ApplicationLogin;

-- Connect to SQL DW database and create a database user
CREATE USER DatabaseUser FOR LOGIN ApplicationLogin;
```

# Access Control - Business requirements

## How do I restrict access to sensitive data to specific database users?

## How do I ensure users only have access to relevant data?

For example, in a hospital only medical staff should be allowed to see patient data that is relevant to them—and not every patient's data.

# Object-level security (tables, views, and more)

## Overview

GRANT controls permissions on designated tables, views, stored procedures, and functions.

Prevent unauthorized queries against certain tables.

Simplifies design and implementation of security at the database level as opposed to application level.

```
-- Grant SELECT permission to user RosaQdM on table Person.Address in the AdventureWorks2012 database
GRANT SELECT ON OBJECT::Person.Address TO RosaQdM;
GO
-- Grant REFERENCES permission on column BusinessEntityID in view HumanResources.vEmployee to user Wanida
GRANT REFERENCES(BusinessEntityID) ON OBJECT::HumanResources.vEmployee to Wanida with GRANT OPTION;
GO
-- Grant EXECUTE permission on stored procedure HumanResources.uspUpdateEmployeeHireInfo to an application role called Recruiting11
USE AdventureWorks2012;
GRANT EXECUTE ON OBJECT::HumanResources.uspUpdateEmployeeHireInfo TO RECRUITING 11;
GO
```

# Row-level security (RLS)

## Overview

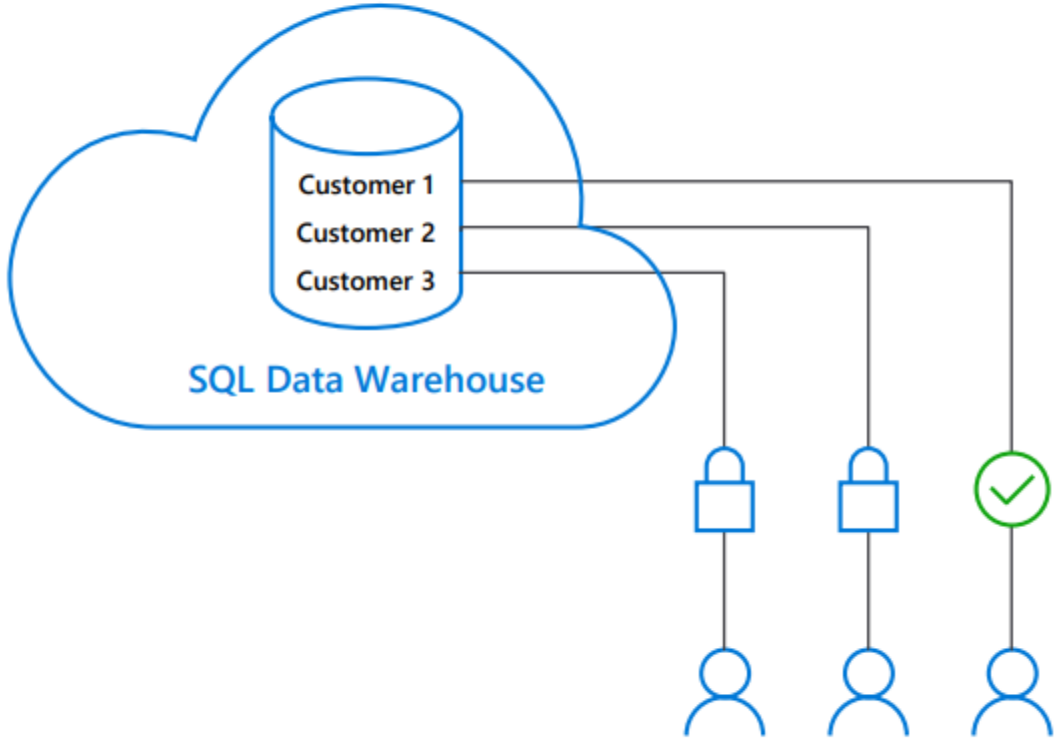
Fine grained access control of specific rows in a database table.

Help prevent unauthorized access when multiple users share the same tables.

Eliminates need to implement connection filtering in multi-tenant applications.

Administer via SQL Server Management Studio or SQL Server Data Tools.

Easily locate enforcement logic inside the database and schema bound to the table.



# Row-level security

## Creating policies

Filter predicates silently filter the rows available to read operations (SELECT, UPDATE, and DELETE).

The following examples demonstrate the use of the CREATE SECURITY POLICY syntax

```
-- The following syntax creates a security policy with a filter predicate for the
Customer table
CREATE SECURITY POLICY [FederatedSecurityPolicy]
ADD FILTER PREDICATE [rls].[fn_securitypredicate]([CustomerId])
ON [dbo].[Customer];


-- Create a new schema and predicate function, which will use the application user ID
stored in CONTEXT_INFO to filter rows.
CREATE FUNCTION rls.fn_securitypredicate (@AppUserId int)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN (
SELECT 1 AS fn_securitypredicate_result
WHERE
DATABASE_PRINCIPAL_ID() = DATABASE_PRINCIPAL_ID('dbo') -- application context
AND CONTEXT_INFO() = CONVERT(VARBINARY(128), @AppUserId));
GO
```

# Row-level security

## Three steps:

1. Policy manager creates filter predicate and security policy in T-SQL, binding the predicate to the patients table.
2. App user (e.g., nurse) selects from Patients table.
3. Security policy transparently rewrites query to apply filter predicate.

Policy manager

Nurse

Application

```
SELECT * FROM Patients
```

Database

Security policy

Patients

Filter Predicate: INNER JOIN...

```
CREATE FUNCTION dbo.fn_securitypredicate(@wing int)
    RETURNS TABLE WITH SCHEMABINDING AS
    return SELECT 1 as [fn_securitypredicate_result] FROM
        StaffDuties d INNER JOIN Employees e
        ON (d.EmpId = e.EmpId)
        WHERE e.UserSID = SUSER_SID() AND @wing = d.Wing;

CREATE SECURITY POLICY dbo.SecPol
    ADD FILTER PREDICATE dbo.fn_securitypredicate(Wing) ON Patients
    WITH (STATE = ON)
```

```
SELECT * FROM Patients
   SEMIJOIN APPLY dbo.fn_securitypredicate(patients.Wing);
```

```
SELECT Patients.* FROM Patients,
   StaffDuties d INNER JOIN Employees e ON (d.EmpId = e.EmpId)
   WHERE e.UserSID = SUSER_SID() AND Patients.wing = d.Wing;
```

# Column-level security

## Overview

Control access of specific columns in a database table based on customer's group membership or execution context.

Simplifies the design and implementation of security by putting restriction logic in database tier as opposed to application tier.

Administer via GRANT T-SQL statement.

Both Azure Active Directory (AAD) and SQL authentication are supported.

# Column-level security

## Three steps:

1. Policy manager creates permission policy in T-SQL, binding the policy to the Patients table on a specific group.
2. App user (for example, a nurse) selects from Patients table.
3. Permission policy prevents access on sensitive data.

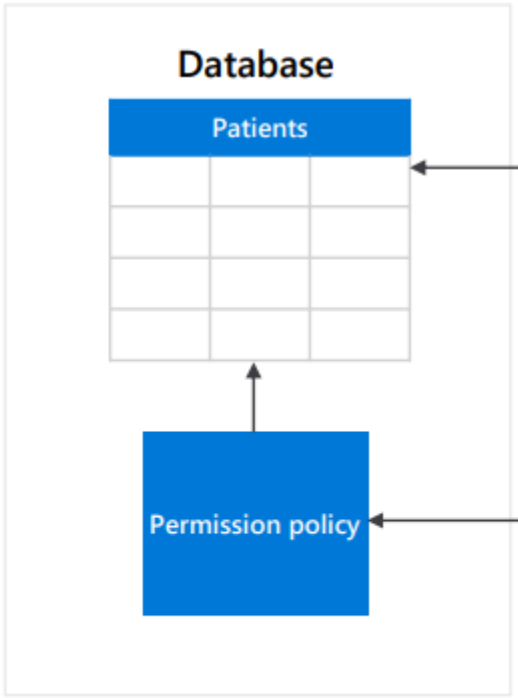
Policy manager

Nurse

Application

```
SELECT * FROM Membership;

Msg 230, Level 14, State 1, Line 12
The SELECT permission was denied on the column
'SSN' of the object 'Membership', database
'CLS_TestDW', schema 'dbo'.
```

Queries executed as 'Nurse' will fail if they include the SSN column




```
CREATE TABLE Patients (
  PatientID int IDENTITY,
  FirstName varchar(100) NULL,
  SSN char(9) NOT NULL,
  LastName varchar(100) NOT NULL,
  Phone varchar(12) NULL,
  Email varchar(100) NULL
);
```

```
GRANT SELECT ON Patients (
  PatientID, FirstName, LastName, Phone, Email
) TO Nurse;
```

Allow 'Nurse' to access all columns except for sensitive SSN column

# Dynamic Data Masking

## Three steps

1. Security officer defines dynamic data masking policy in T-SQL over sensitive data in the Employee table. The security officer uses the built-in masking functions (default, email, random)

2. The app-user selects from the Employee table

3. The dynamic data masking policy obfuscates the sensitive data in the query results for non-privileged users

Security officer

1

```
ALTER TABLE [Employee]
ALTER COLUMN [SocialSecurityNumber]
ADD MASKED WITH (FUNCTION = 'DEFAULT()')

ALTER TABLE [Employee]
ALTER COLUMN [Email]
ADD MASKED WITH (FUNCTION = 'EMAIL()')

ALTER TABLE [Employee]
ALTER COLUMN [Salary]
ADD MASKED WITH (FUNCTION = 'RANDOM(1,20000)')

GRANT UNMASK to admin1
```

2

Business app

```
SELECT [First Name],
       [Social Security Number],
       [Email],
       [Salary]
FROM   [Employee]
```

3

Non-masked data (admin login)

| | First Name | Social Security Num... | Email | Salary |
|---|---|---|---|---|
| 1 | LILA | 758-10-9637 | lila.bamett@comcast.net | 1012794 |
| 2 | JAMIE | 113-29-4314 | jamie.brown@ntlworld.com | 1025713 |
| 3 | SHELLEY | 550-72-2028 | shelley.lynn@charter.net | 1040131 |
| 4 | MARCELLA | 903-94-5665 | marcella.estrada@comcast.net | 1040753 |
| 5 | GILBERT | 376-79-4787 | gilbert.juarez@verizon.net | 1041308 |

Masked data (admin1 login)

| | First Name | Social Security Number | Email | Salary |
|---|---|---|---|---|
| 1 | LILA | XXX-XX-XX37 | lXX@XXXX.net | 8940 |
| 2 | JAMIE | XXX-XX-XX14 | jXX@XXXX.com | 19582 |
| 3 | SHELLEY | XXX-XX-XX28 | sXX@XXXX.net | 3713 |
| 4 | MARCELLA | XXX-XX-XX65 | mXX@XXXX.net | 11572 |
| 5 | GILBERT | XXX-XX-XX87 | gXX@XXXX.net | 4487 |

# Database Views

Views

Materialized Views

# Materialized views

## Overview

A materialized view pre-computes, stores, and maintains its data like a table.

Materialized 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 Azure Synapse Analytics 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

## Benefits

Automatic and synchronous data refresh with data changes in base tables. No user action is required.

High availability and resiliency as regular tables

```
-- Create indexed view
CREATE MATERIALIZED 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;
```

# Materialized views - example

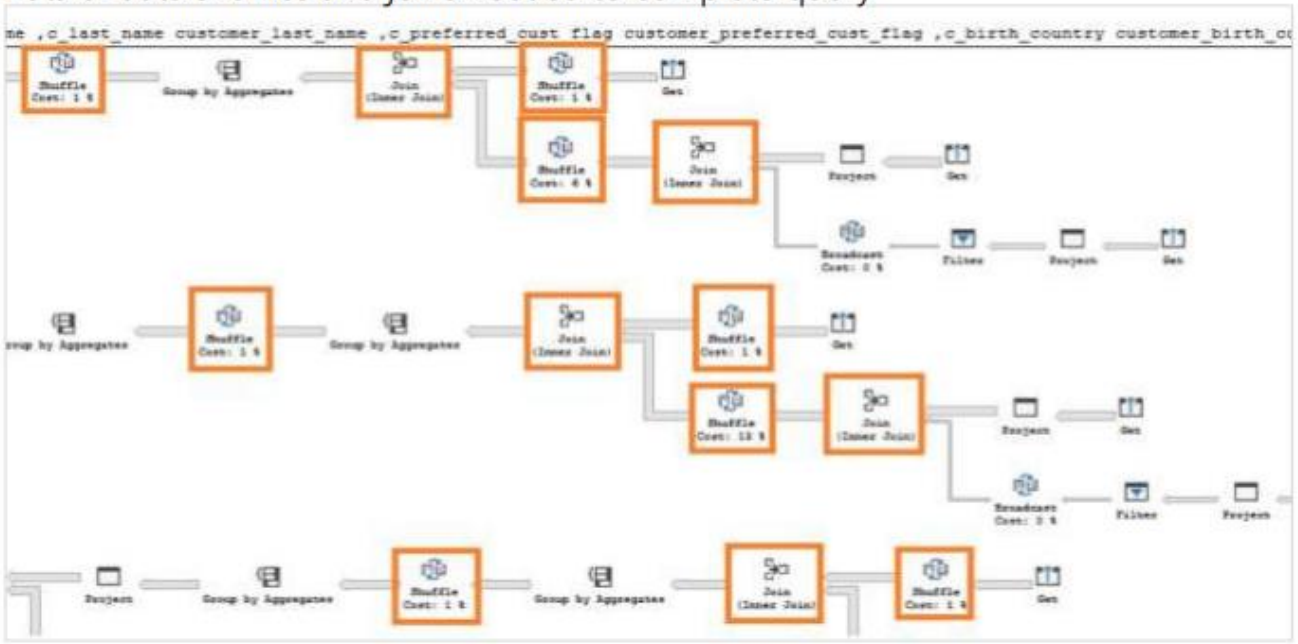
In this example, a query to get the year total sales per customer is shown to have a lot of data shuffles and joins that contribute to slow performance:

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



# 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
```

# COPY command

## Overview

Copies data from source to destination

## Benefits

Retrieves data from all files from the folder and all its subfolders.

Supports multiple locations from the same storage account, separated by comma

Supports Azure Data Lake Storage (ADLS) Gen 2 and Azure Blob Storage.

Supports CSV, PARQUET, ORC file formats

```
COPY INTO test_1
FROM
'https://XXX.blob.core.windows.net/customerdatasets/test_1.txt'
WITH (
    FILE_TYPE = 'CSV',
    CREDENTIAL=(IDENTITY= 'Shared Access Signature',
SECRET='<Your_SAS_Token>'),
    FIELDQUOTE = '"',
    FIELDTERMINATOR=';',
    ROWTERMINATOR='0X0A',
    ENCODING = 'UTF8',
    DATEFORMAT = 'ymd',
    MAXERRORS = 10,
    ERRORFILE = '/errorsfolder/'--path starting from
the storage container,
    IDENTITY_INSERT
    )
```

```
COPY INTO test_parquet
FROM
'https://XXX.blob.core.windows.net/customerdatasets/test
.parquet'
WITH (
    FILE_FORMAT = myFileFormat
    CREDENTIAL=(IDENTITY= 'Shared Access Signature',
SECRET='<Your_SAS_Token>')
)
```

# Dynamic Management Views (DMVs)

## Overview

Dynamic Management Views (DMV) are queries that return information about model objects, server operations, and server health.

## Benefits:

Simple SQL syntax

Returns result in table format

Easier to read and copy result

# SQL Monitor with DMVs

## Overview

Offers monitoring of

-all open, closed sessions

-count sessions by user

-count completed queries by user

-all active, complete queries

-longest running queries

-memory consumption

Count sessions by user

```
--count sessions by user
SELECT login_name, COUNT(*) as session_count FROM
sys.dm_pdw_exec_sessions where status = 'Closed' and session_id
<> session_id() GROUP BY login_name;
```

List all open sessions

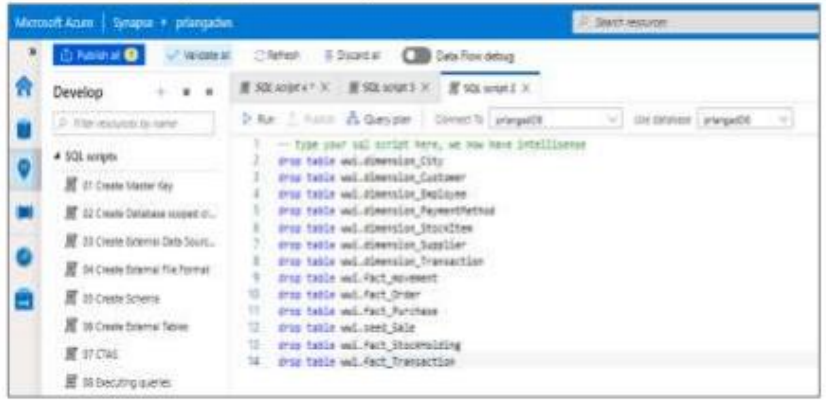
```
-- List all open sessions
SELECT * FROM sys.dm_pdw_exec_sessions where status <> 'Closed'
and session_id <> session_id();
```

List all active queries

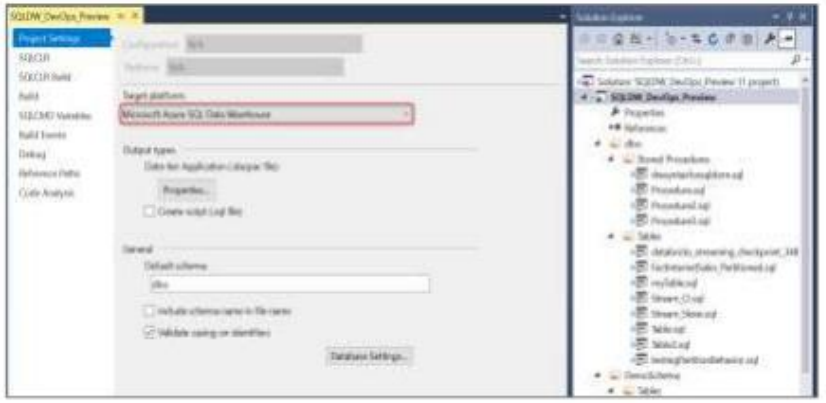
```
-- List all active queries
SELECT * FROM sys.dm_pdw_exec_requests WHERE status not in
('Completed','Failed','Cancelled') AND session_id <> session_id()
ORDER BY submit_time DESC;
```

# Developer Tools

*Azure Synapse Analytics*

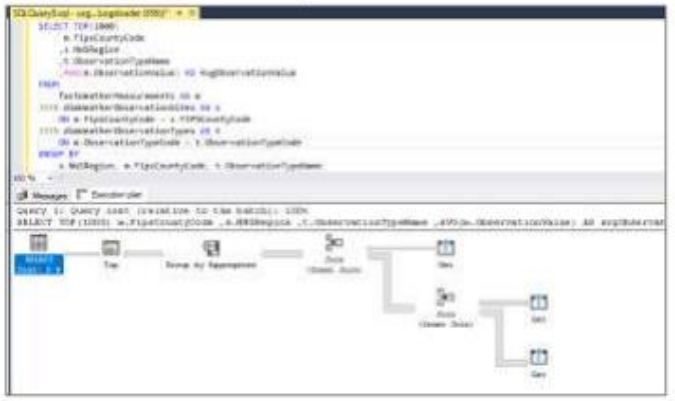


*Visual Studio - SSDT database projects*

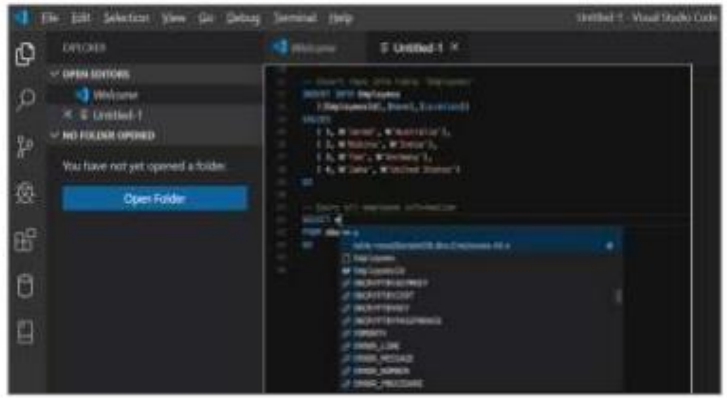


*Azure Data Studio (queries, extensions etc.)*

*SQL Server Management Studio (queries, execution plans etc.)*



*Visual Studio Code*



# Continuous integration and delivery (CI/CD)

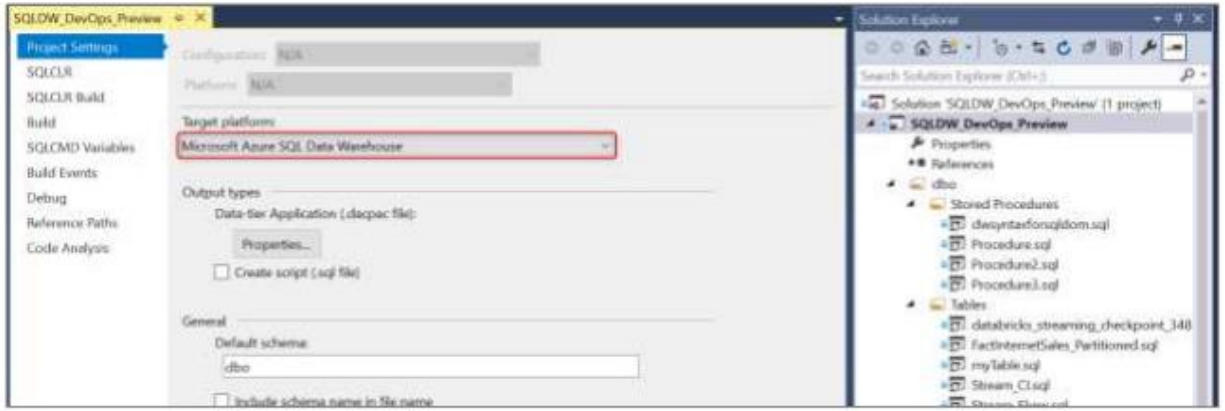
## Overview

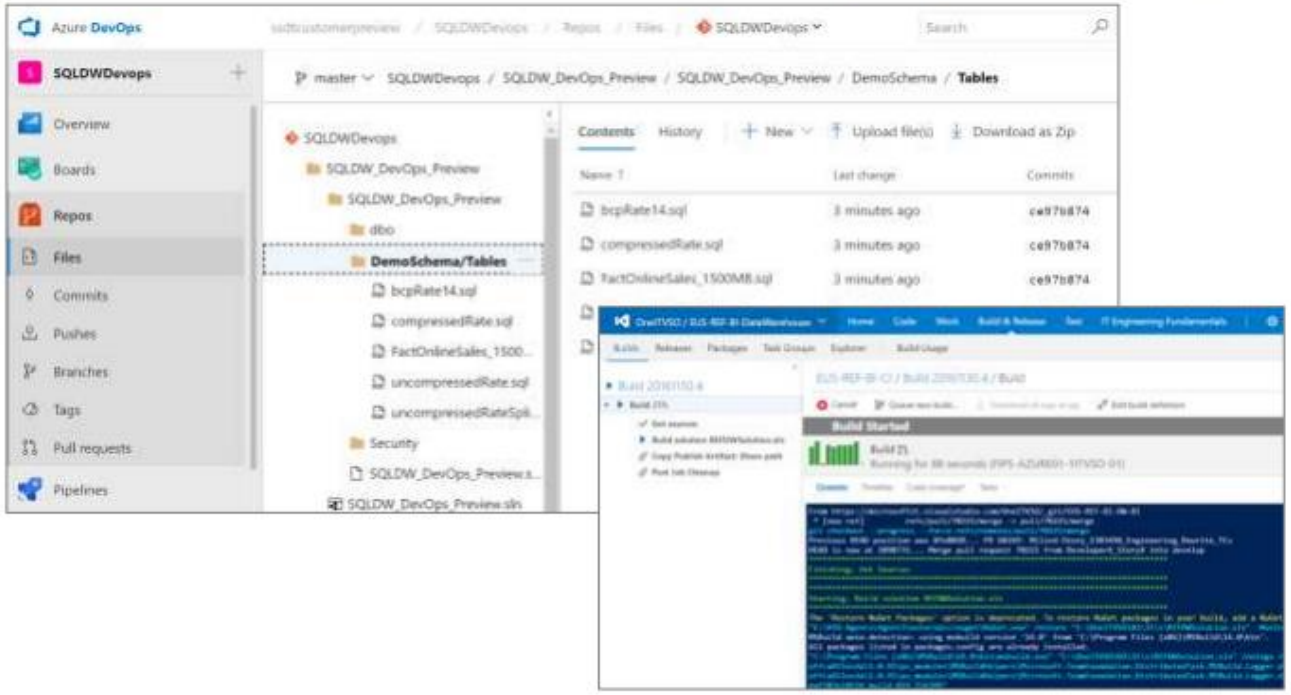
Database project support in SQL Server Data Tools (SSDT) allows teams of developers to collaborate over a version-controlled data warehouse, and track, deploy and test schema changes.

## Benefits

Database project support includes first-class integration with Azure DevOps. This adds support for:

- **Azure Pipelines** to run CI/CD workflows for any platform (Linux, macOS, and Windows)
- **Azure Repos** to store project files in source control
- **Azure Test Plans** to run automated check-in tests to verify schema updates and modifications
- Growing ecosystem of third-party integrations that can be used to complement existing workflows (Timetracker, Microsoft Teams, Slack, Jenkins, etc.)



# Azure Advisor recommendations

## Suboptimal Table Distribution

Reduce data movement by replicating tables

## Data Skew

Choose new hash-distribution key

Slowest distribution limits performance

## Cache Misses

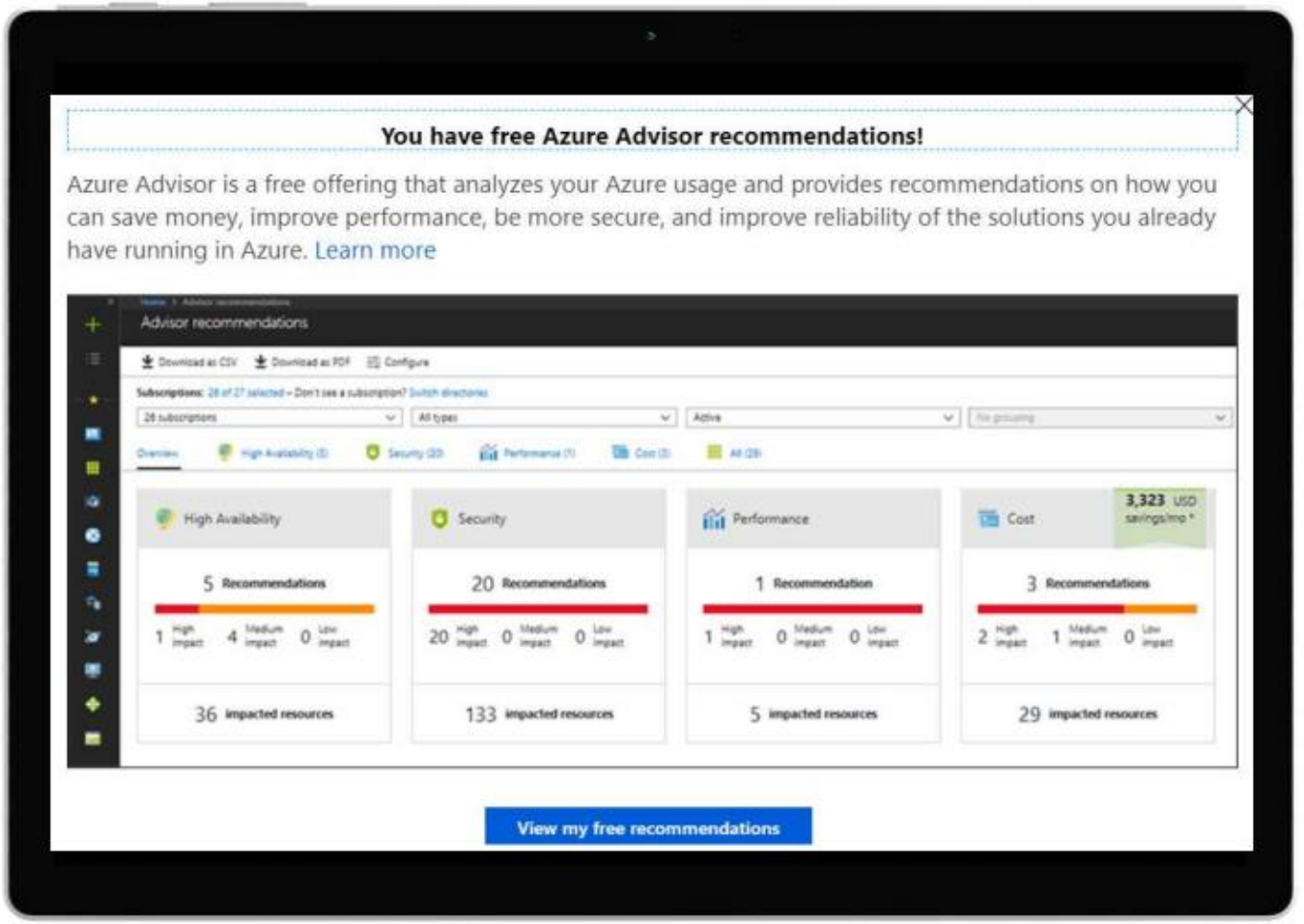
Provision additional capacity

## Tempdb Contention

Scale or update user resource class

## Suboptimal Plan Selection

Create or update table statistics



# Maintenance windows

## Overview

Choose a time window for your upgrades.

Select a primary and secondary window within a seven-day period.

Windows can be from 3 to 8 hours.

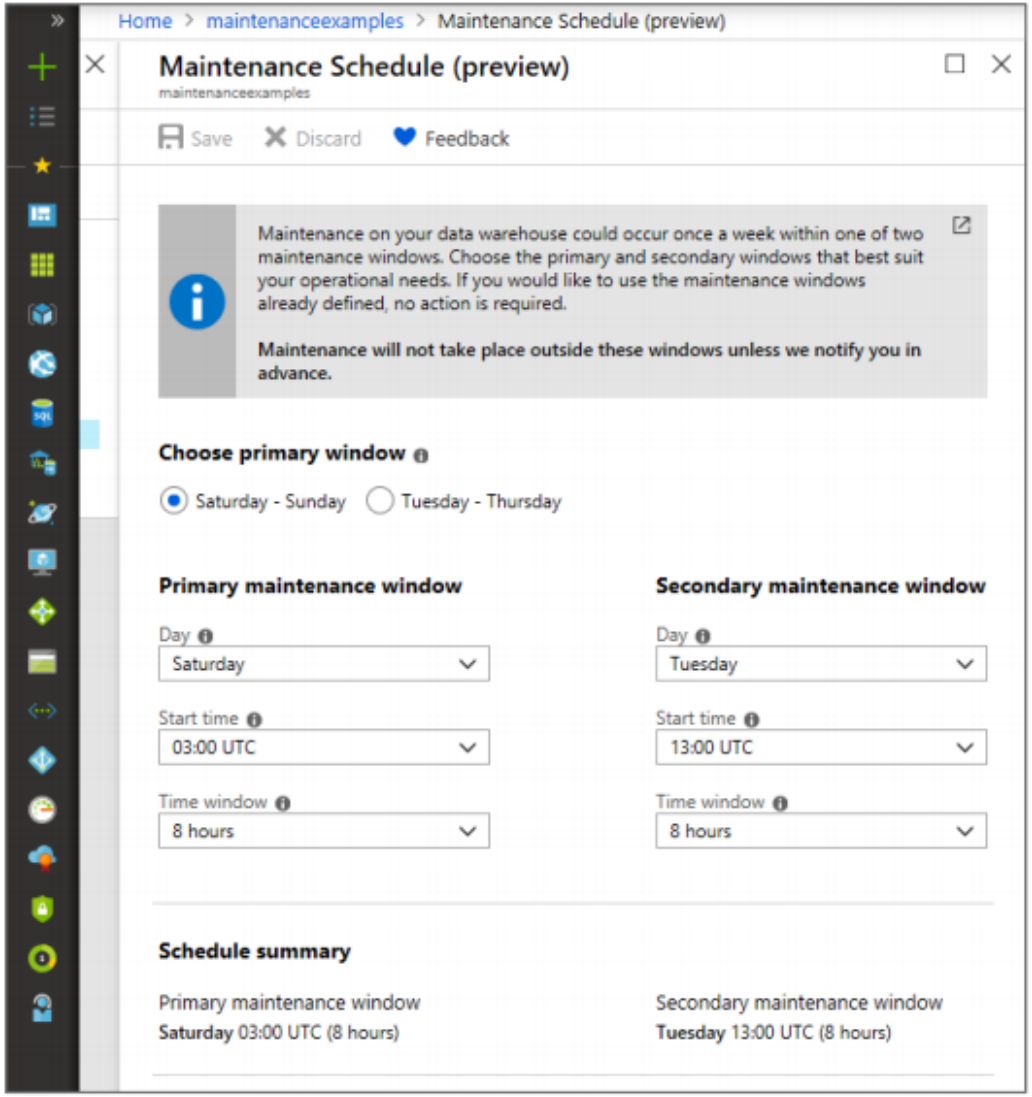
24-hour advance notification for maintenance events.

## Benefits

Ensure upgrades happen on your schedule.

Predictable planning for long-running jobs.

Stay informed of start and end of maintenance.



# Automatic statistics management

## Overview

Statistics are automatically created and maintained for SQL pool. Incoming queries are analyzed, and individual column statistics are generated on the columns that improve cardinality estimates to enhance query performance.

Statistics are automatically updated as data modifications occur in underlying tables. By default, these updates are synchronous but can be configured to be asynchronous.

Statistics are considered out of date when:

- There was a data change on an empty table
- The number of rows in the table at time of statistics creation was 500 or less, and more than 500 rows have been updated
- The number of rows in the table at time of statistics creation was more than 500, and more than 500 + 20% of rows have been updated

```
-- Turn on/off auto-create statistics settings
ALTER DATABASE {database_name}
SET AUTO_CREATE_STATISTICS { ON | OFF }

-- Turn on/off auto-update statistics settings
ALTER DATABASE {database_name}
SET AUTO_UPDATE_STATISTICS { ON | OFF }

-- Configure synchronous/asynchronous update
ALTER DATABASE {database_name}
SET AUTO_UPDATE_STATISTICS_ASYNC { ON | OFF }

-- Check statistics settings for a database
SELECT      is_auto_create_stats_on,
            is_auto_update_stats_on,
            is_auto_update_stats_async_on
FROM        sys.databases
```

# Distributed Query Processor (DQP)

- **Auto-scale compute nodes** - Instruct the underlying fabric the need for more compute power to adjust to peaks during the workload. If compute power is granted, the Polaris DQP will re-distribute tasks leveraging the new compute container. Note that in-flight tasks in the previous topology continue running, while new queries get the new compute power with the new re-balancing
- **Compute node fault tolerance** - Recover from faulty nodes while a query is running. If a node fails the DQP re-schedules the tasks in the faulted node through the remainder of the healthy topology
- **Compute node hot spot: rebalance queries or scale out nodes** - Can detect hot spots in the existing topology. That is, overloaded compute nodes due to data skew. In the advent of a compute node running hot because of skewed tasks, the DQP can decide to re-schedule some of the tasks assigned to that compute node amongst others where the load is less
- **Multi-cluster** - Multiple compute pools accessing the same data
- **Cross-database queries** – A query can specify multiple databases

These features work for both on-demand and provisioned over ADLS Gen2 and relational databases

# Synapse SQL on-demand scenarios

| | |
|---|---|
| **Discovery and exploration** | What's in this file? How many rows are there? What's the max value?<br><br>**SQL On-demand reduces data lake exploration to the right-click!** |
| **Data transformation** | How to convert CSVs to Parquet quickly? How to transform the raw data?<br><br>**Use the full power of T-SQL to transform the data in the data lake** |

# SQL On-Demand

## Overview

An interactive query service that provides T-SQL queries over high scale data in Azure Storage.

## Benefits

Serverless

No infrastructure

***Pay only for query execution***

No ETL

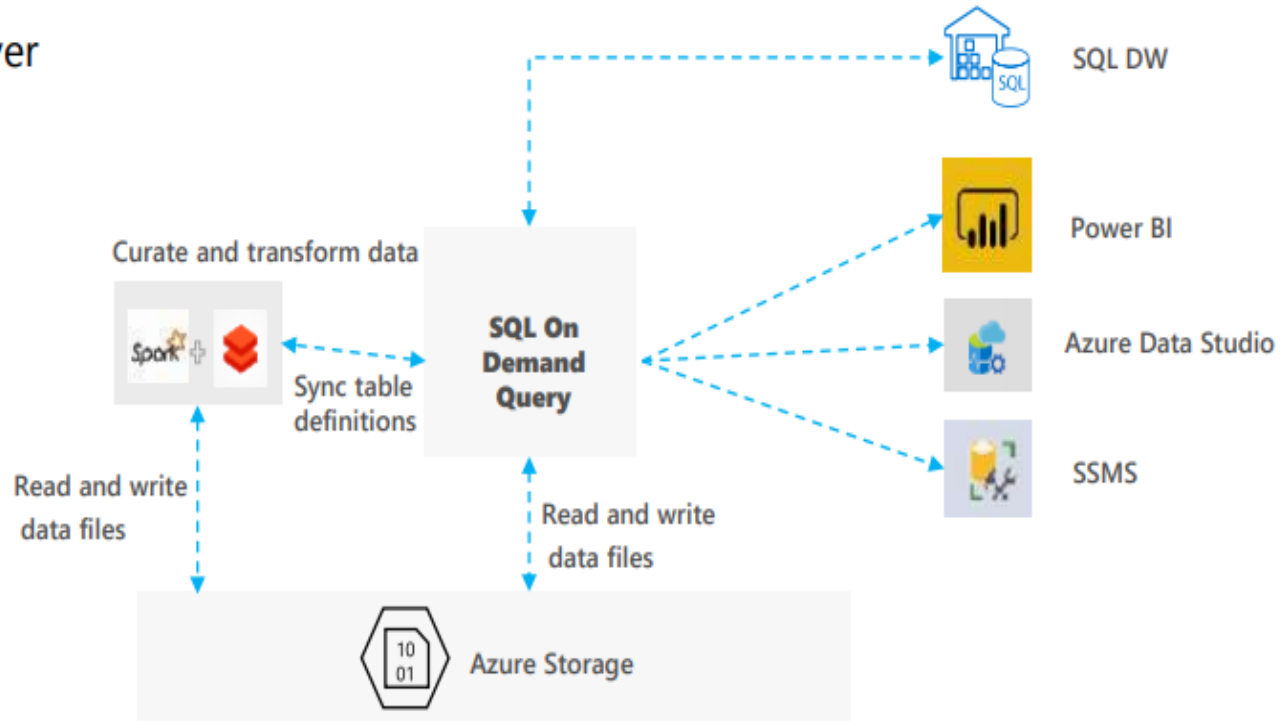
Offers security

Data integration with Databricks, HDInsight

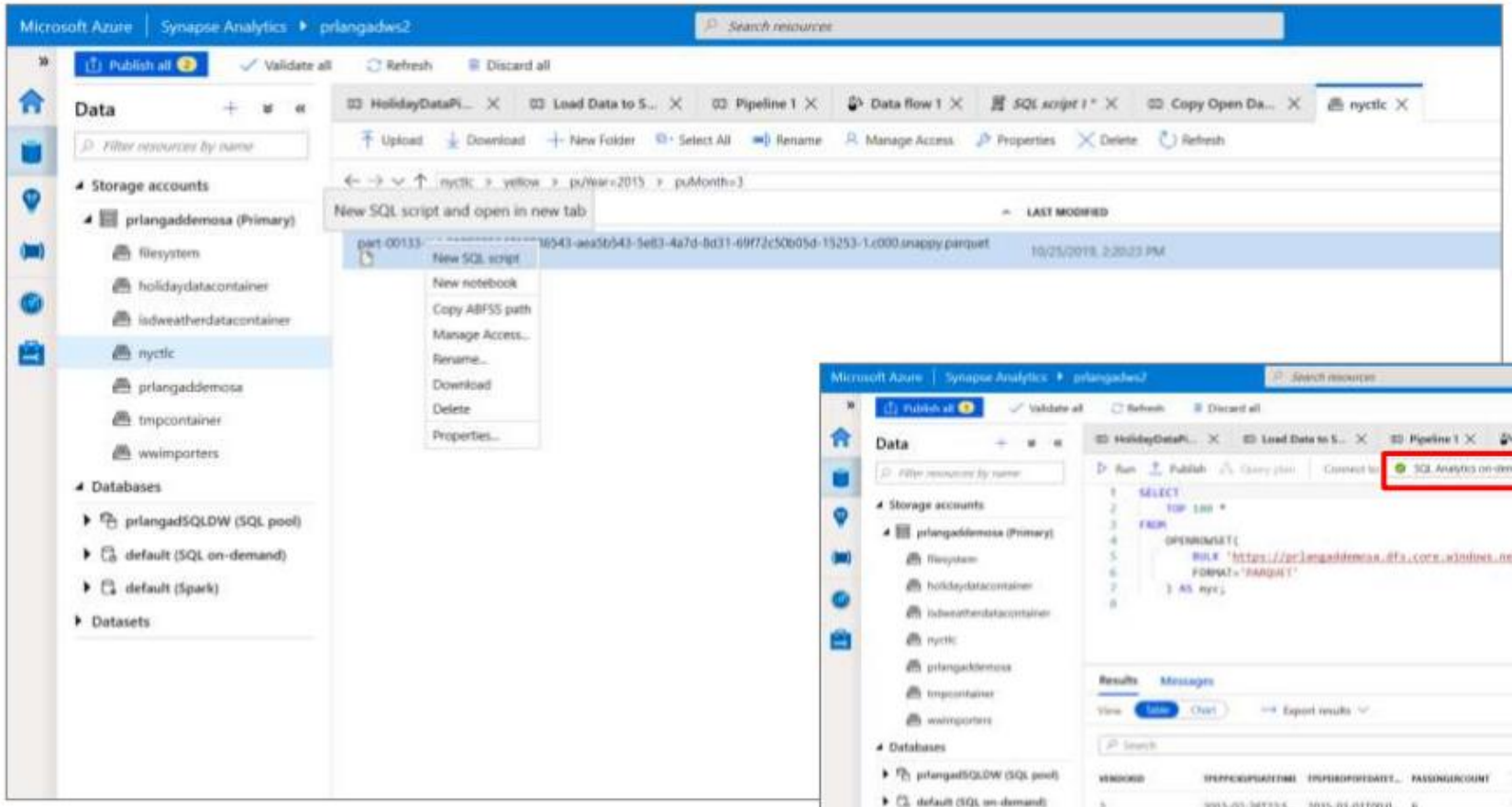
T-SQL syntax to query data

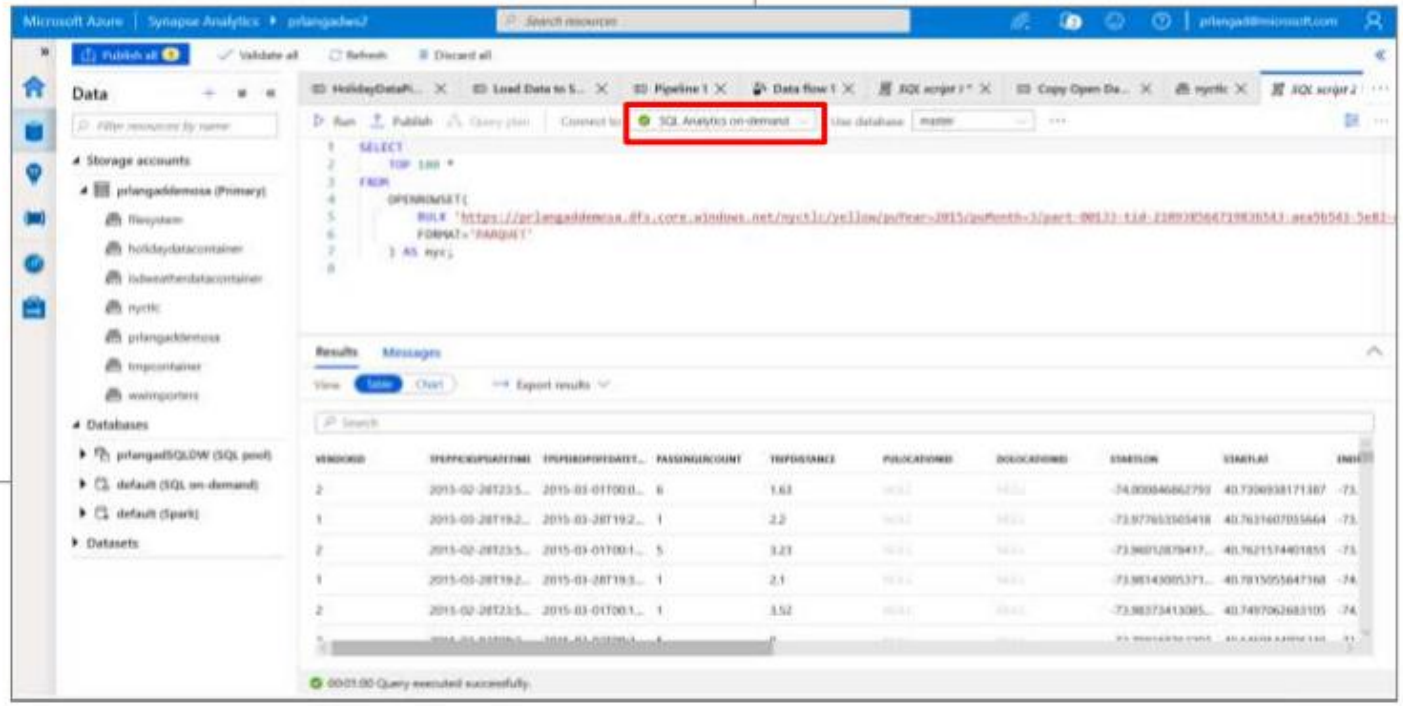
Supports data in various formats (Parquet, CSV, JSON)

Support for BI ecosystem



# SQL On Demand – Querying on storage



# SQL On Demand – Querying CSV File

## Overview

Uses OPENROWSET function to access data

## Benefits

Ability to read CSV File with

- no header row, Windows style new line
- no header row, Unix-style new line
- header row, Unix-style new line
- header row, Unix-style new line, quoted
- header row, Unix-style new line, escape
- header row, Unix-style new line, tab-delimited
- without specifying all columns

```
SELECT *
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/population/populat
ion.csv',
        FORMAT = 'CSV',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '\n'
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017
```

| | country_code | country_name | year | population |
|---|---|---|---|---|
| 1 | LU | Luxembourg | 2017 | 594130 |

# SQL On Demand – Querying CSV File

Read CSV file - header row, Unix-style new line

```
SELECT *
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/population-
unix-hdr/population.csv',
        FORMAT = 'CSV',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '0x0a',
        FIRSTROW = 2
    )
    WITH (
        [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
        [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
        [year] smallint,
        [population] bigint
    ) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017
```

| | country_code | country_name | year | population |
|---|---|---|---|---|
| 1 | LU | Luxembourg | 2017 | 594130 |

Read CSV file - without specifying all columns

```
SELECT
    COUNT(DISTINCT country_name) AS countries
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/popul
ation/population.csv',
        FORMAT = 'CSV',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '\n'
    )
WITH (
        [country_name] VARCHAR (100) COLLATE Latin1_Gener
al_BIN2 2
) AS [r]
```

| | countries |
|---|---|
| 1 | 228 |

# SQL On Demand – Querying folders

## Overview

Uses OPENROWSET function to access data from multiple files or folders

## Benefits

Offers reading multiple files/folders through usage of wildcards

Offers reading specific file/folder

Supports use of multiple wildcards

```
SELECT YEAR(pickup_datetime) as [year], SUM(passenger_count) AS
passengers_total, COUNT(*) AS [rides_total]
FROM OPENROWSET(
BULK 'https://XXX.blob.core.windows.net/csv/taxi/*.*',
FORMAT = 'CSV'
, FIRSTROW = 2 )
WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        rate_code INT,
        store_and_fwd_flag VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_location_id INT,
        dropoff_location_id INT,
        payment_type INT,
        fare_amount FLOAT,
        extra FLOAT, mta_tax FLOAT,
        tip_amount FLOAT,
        tolls_amount FLOAT,
        improvement_surcharge FLOAT,
        total_amount FLOAT
        ) AS nyc
GROUP BY YEAR(pickup_datetime)
ORDER BY YEAR(pickup_datetime)
```

| | year | passengers_total | rides_total |
|---|---|---|---|
| 1 | 2001 | 14 | 10 |
| 2 | 2002 | 29 | 16 |
| 3 | 2003 | 22 | 16 |
| 4 | 2008 | 378 | 188 |
| 5 | 2009 | 594 | 353 |
| 6 | 2016 | 102093687 | 61758523 |
| 7 | 2017 | 184464988 | 113496932 |
| 8 | 2018 | 86272771 | 53925040 |
| 9 | 2019 | 37 | 29 |
| … | 2020 | 6 | 6 |

# SQL On Demand – Querying folders

## Read all files from multiple folders

```
SELECT YEAR(pickup_datetime) as [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM OPENROWSET(
    BULK 'https://XXX.blob.core.windows.net/csv/t*i/',
        FORMAT = 'CSV',
        FIRSTROW = 2     )
    WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        <… columns>
    ) AS nyc
GROUP BY YEAR(pickup_datetime)
ORDER BY YEAR(pickup_datetime)
```

| | year | passengers_total | rides_total |
|---|---|---|---|
| 1 | 2001 | 14 | 10 |
| 2 | 2002 | 29 | 16 |
| 3 | 2003 | 22 | 16 |
| 4 | 2008 | 378 | 188 |
| 5 | 2009 | 594 | 353 |
| 6 | 2016 | 102093687 | 61758523 |
| 7 | 2017 | 184464988 | 113496932 |
| 8 | 2018 | 86272771 | 53925040 |
| 9 | 2019 | 37 | 29 |
| … | 2020 | 6 | 6 |

## Read subset of files in folder

```
SELECT
    payment_type,
    SUM(fare_amount) AS fare_total
FROM OPENROWSET(
    BULK 'https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_2017-*.csv',
        FORMAT = 'CSV',
        FIRSTROW = 2     )
    WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        <…columns>
    ) AS nyc
GROUP BY payment_type
ORDER BY payment_type
```

| | payment_type | fare_total |
|---|---|---|
| 1 | 1 | 1026072325.579… |
| 2 | 2 | 441093322.8000… |
| 3 | 3 | 10435183.04 |
| 4 | 4 | 3304550.99 |
| 5 | 5 | 14 |

# SQL On Demand – Querying specific files

## Overview

filename – Provides file name that originates row result

filepath – Provides full path when no parameter is passed or part of path  when parameter is passed that originates result

## Benefits

Provides source name/path of file/folder for row result set

Example of filename function

```
SELECT
    r.filename() AS [filename]
    ,COUNT_BIG(*) AS [rows]
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_201
7-1*.csv',
        FORMAT = 'CSV',
        FIRSTROW = 2
    )
    WITH (
        vendor_id INT,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count SMALLINT,
        trip_distance FLOAT,
        <…columns>
    ) AS [r]

GROUP BY r.filename()

ORDER BY [filename]
```

| | filename | rows |
|---|---|---|
| 1 | yellow_tripdata_2017-10.csv | 9768815 |
| 2 | yellow_tripdata_2017-11.csv | 9284803 |
| 3 | yellow_tripdata_2017-12.csv | 9508276 |

# SQL On Demand – Querying specific files

Example of filepath function

```
SELECT
    r.filepath() AS filepath
    ,r.filepath(1) AS [year]
    ,r.filepath(2) AS [month]
    ,COUNT_BIG(*) AS [rows]
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_*-*.csv',
        FORMAT = 'CSV',
        FIRSTROW = 2    )
WITH (
    vendor_id INT,
    pickup_datetime DATETIME2,
    dropoff_datetime DATETIME2,
    passenger_count SMALLINT,
    trip_distance FLOAT,
   <… columns>
) AS [r]

WHERE r.filepath(1) IN ('2017')
  AND r.filepath(2) IN ('10', '11', '12')

GROUP BY    r.filepath() ,r.filepath(1) ,r.filepath(2)
ORDER BY    filepath
```

| filepath | year | month | rows |
|---|---|---|---|
| https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_2017-10.csv | 2017 | 10 | 9768815 |
| https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_2017-11.csv | 2017 | 11 | 9284803 |
| https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_2017-12.csv | 2017 | 12 | 9508276 |

# SQL On Demand – Querying Parquet files

## Overview

Uses OPENROWSET function to access data

## Benefits

Ability to specify column names of interest

Offers auto reading of column names and data types

Provides target specific partitions using filepath function

```
SELECT
        YEAR(pickup_datetime),
        passenger_count,
        COUNT(*) AS cnt
FROM
    OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/parquet/taxi/*/*/*',
        FORMAT='PARQUET'
    ) WITH (
        pickup_datetime DATETIME2,
        passenger_count INT
    ) AS nyc
GROUP BY
    passenger_count,
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime),
    passenger_count
```

| | (No column name) | passenger_count | cnt |
|---|---|---|---|
| 1 | 2016 | 0 | 2557 |
| 2 | 2016 | 1 | 43735845 |
| 3 | 2016 | 2 | 9056714 |
| 4 | 2016 | 3 | 2610541 |
| 5 | 2016 | 4 | 1309639 |
| 6 | 2016 | 5 | 3086097 |
| 7 | 2016 | 6 | 1956607 |

# SQL On Demand – Creating views

## Overview

Create views using SQL On Demand queries

## Benefits

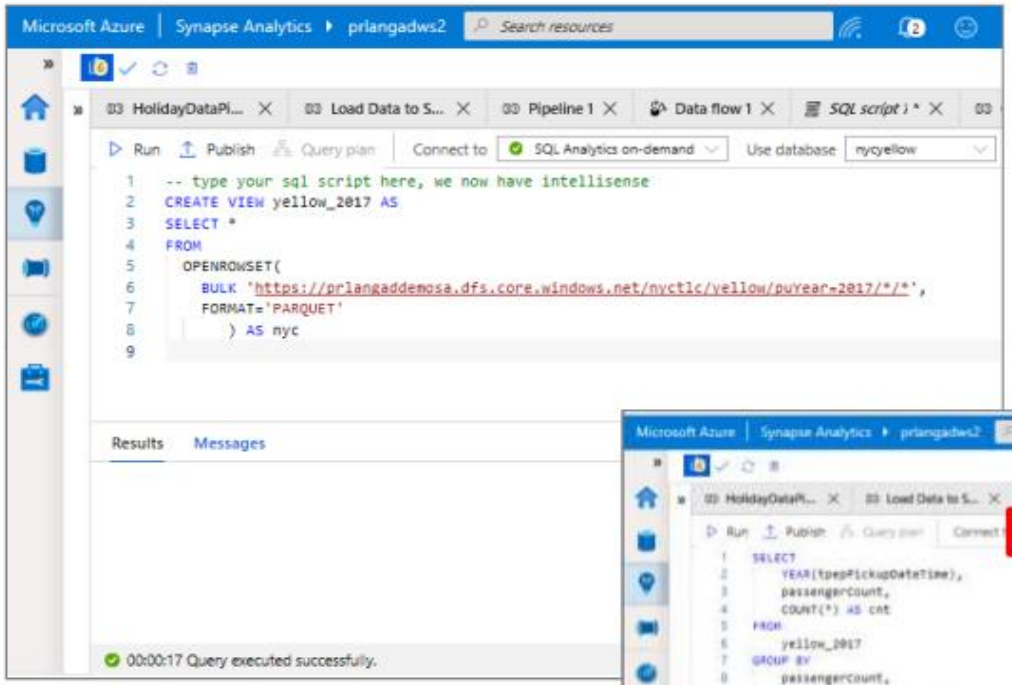
Works same as standard views

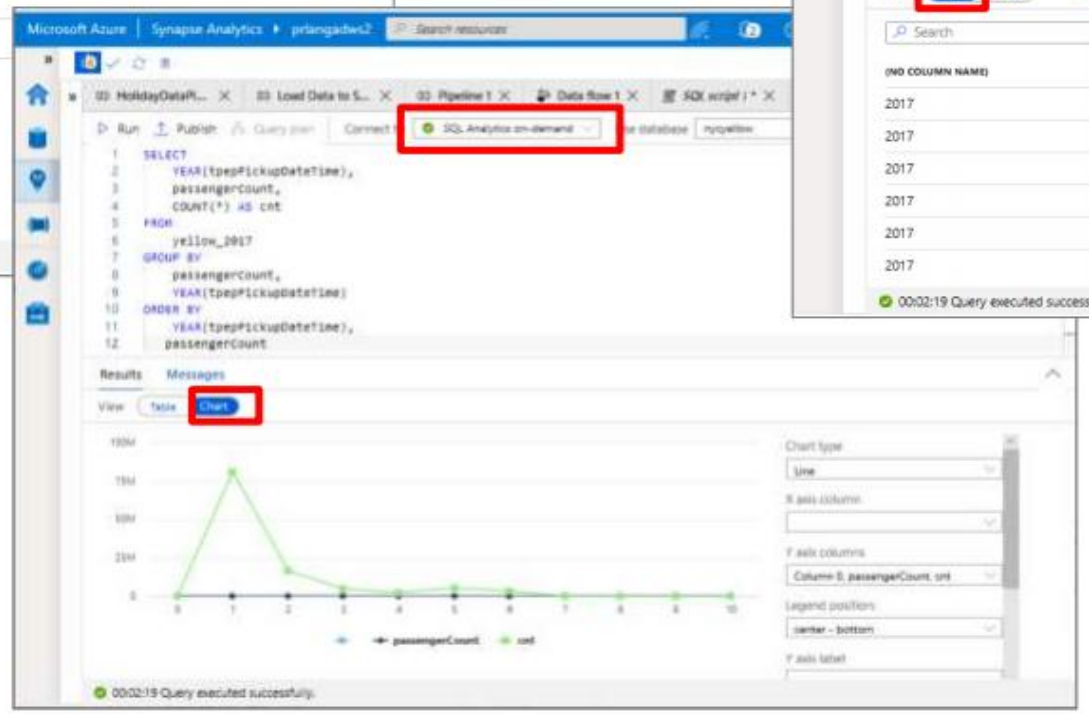
```
USE [mydbname]
GO

IF EXISTS(select * FROM sys.views where name = 'populationView')
DROP VIEW populationView
GO

CREATE VIEW populationView AS
SELECT *
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/population/population.csv',
        FORMAT = 'CSV',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '\n'
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r]
```

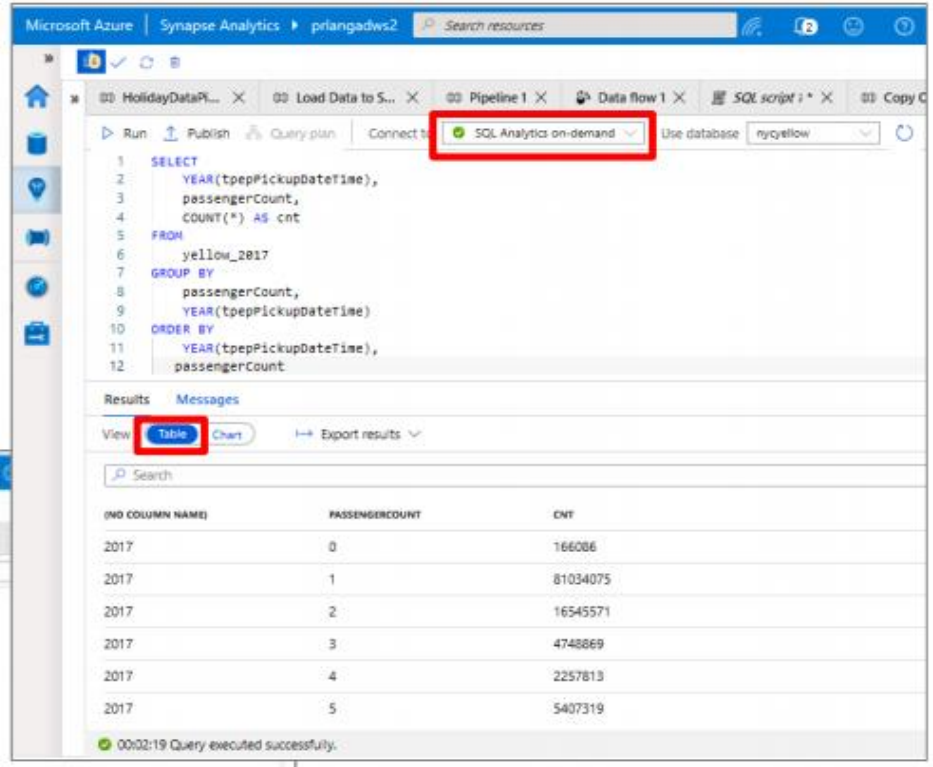
```
SELECT
    country_name, population
FROM populationView
WHERE
    [year] = 2019
ORDER BY
    [population] DESC
```

| | country_name | population |
|---|---|---|
| 1 | China | 1389618778 |
| 2 | India | 1311559204 |
| 3 | United States | 331883986 |
| 4 | Indonesia | 264935824 |
| 5 | Pakistan | 210797836 |
| 6 | Brazil | 210301591 |
| 7 | Nigeria | 208679114 |
| 8 | Bangladesh | 161062905 |
| 9 | Russia | 141944641 |
| 10 | Mexico | 127318112 |

# SQL On Demand – Creating views



# SQL On Demand – Querying JSON files

## Overview

Read JSON files and provides data in tabular format

## Benefits

Supports OPENJSON, JSON_VALUE and JSON_QUERY functions

```
SELECT *
FROM
    OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/json/books/book
1.json',
        FORMAT='CSV',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b',
        ROWTERMINATOR = '0x0b'
    )
    WITH (
        jsonContent varchar(8000)
    ) AS [r]
```

| | jsonContent |
|---|---|
| 1 | {"_id": "kim95", "type": "Book", "title": "Modern Databas… |

# SQL On Demand – Querying JSON files

Example of JSON_VALUE function

```
SELECT
    JSON_VALUE(jsonContent, '$.title') AS title,
    JSON_VALUE(jsonContent, '$.publisher') as publisher,
    jsonContent
FROM
    OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/json/books/*.json',
        FORMAT='CSV',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b',
        ROWTERMINATOR = '0x0b'
    )
    WITH (
        jsonContent varchar(8000)
    ) AS [r]
WHERE
    JSON
cal Meth
```

| | title | publisher | jsonContent |
|---|---|---|---|
| 1 | Probabilistic and Statistical Methods in Cryptology, An Int… | Springer | {"_id": "neuen… |

Example of JSON_QUERY function

```
SELECT
    JSON_QUERY(jsonContent, '$.authors') AS authors,
    jsonContent
FROM
    OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/json/books/*.json',
        FORMAT='CSV',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b',
        ROWTERMINATOR = '0x0b'
    )
    WITH (
        jsonContent varchar(8000)
    ) AS [r]
WHERE
    JSON_VALUE(jsonContent, '$.title') = 'Probabilistic and Statist
ical Methods in Cryptology, An Introduction by Selected Topics'
```

| | authors | jsonContent |
|---|---|---|
| 1 | ["Daniel Neuenschwander"] | {"_id": "neuenschwander04", "type": "Book", "title": "Probabi… |

# Create External Table As Select

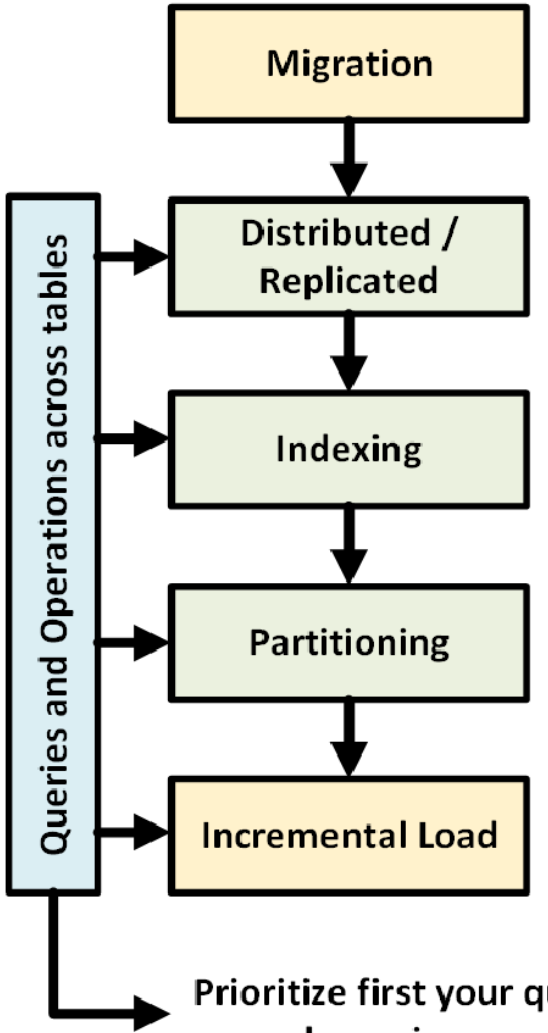
## Overview

Creates an external table and then exports results of the Select statement. These operations will import data into the database for the duration of the query

Steps:

1. Create Master Key
2. Create Credentials
3. Create External Data Source
4. Create External Data Format
5. Create External Table

```
-- Create a database master key if one does not already exist
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
)
-- Create an external file format
CREATE EXTERNAL FILE FORMAT MyAzureCSVFormat
WITH (FORMAT_TYPE = DELIMITEDTEXT,
      FORMAT_OPTIONS(
          FIELD_TERMINATOR = ',',
          FIRST_ROW = 2)
--Create an external table
CREATE EXTERNAL TABLE dbo.FactInternetSalesNew
WITH(
        LOCATION = '/files/Customer',
        DATA_SOURCE = MyAzureStorage,
        FILE_FORMAT = MyAzureCSVFormat
    )
AS SELECT T1.* FROM dbo.FactInternetSales T1 JOIN dbo.DimCustomer T2
ON ( T1.CustomerKey = T2.CustomerKey )
OPTION ( HASH JOIN );
```




Load your data as fast as possible to a staging table

Limit data movement when you “Group By” and “Join” tables

Optimize your table for read performance with the right indexing

Improved performance when you “Filter” on the partition key and help manage the data lifecycle

Minimize the disruption for your business users

**Prioritize first your queries and transformations in dedicated SQL pool, second your incremental loads**