DB Design and Tuning for Azure Synapse DB

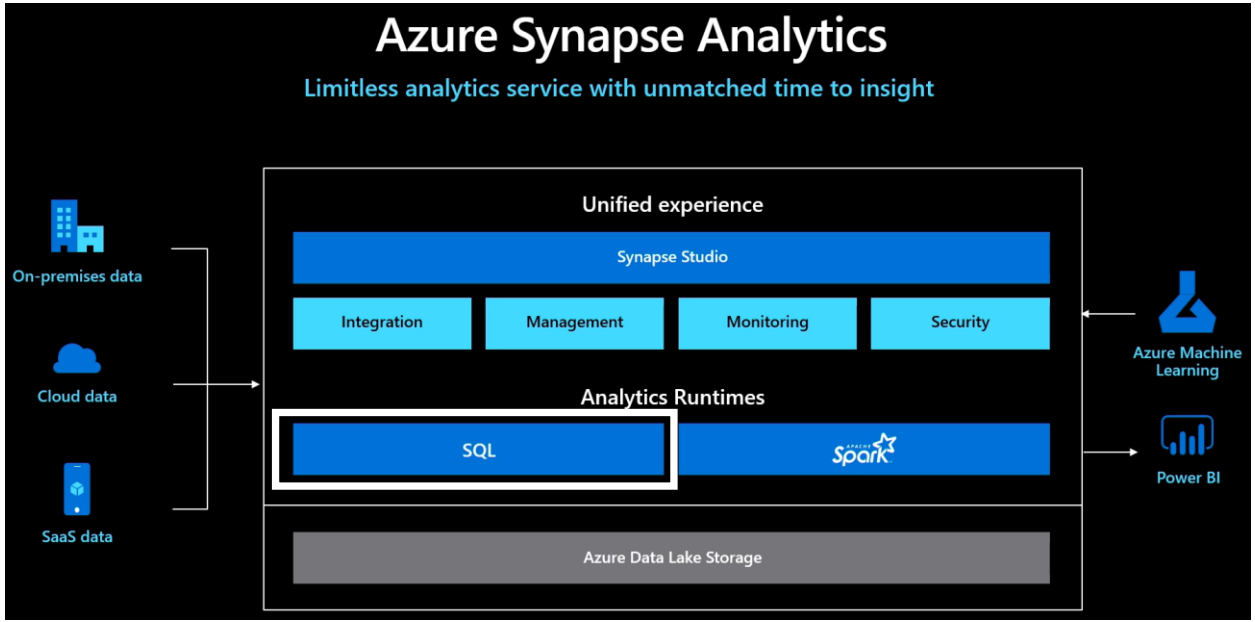
Ted.Tasker@Insight.com
Central Ohio Azure User Group

Insight Presentation

Agenda

- Azure SQL DW to Synapse Analytics
- DB Massive Parallel Processing (MPP) Architecture
- MPP Execution
- Fundamentals: Table Distribution Types and Storage
- Distribution Keys and Skew
- Coding Best Practices
- Questions

Azure Synapse Analytics



SQL box = Azure SQL DW

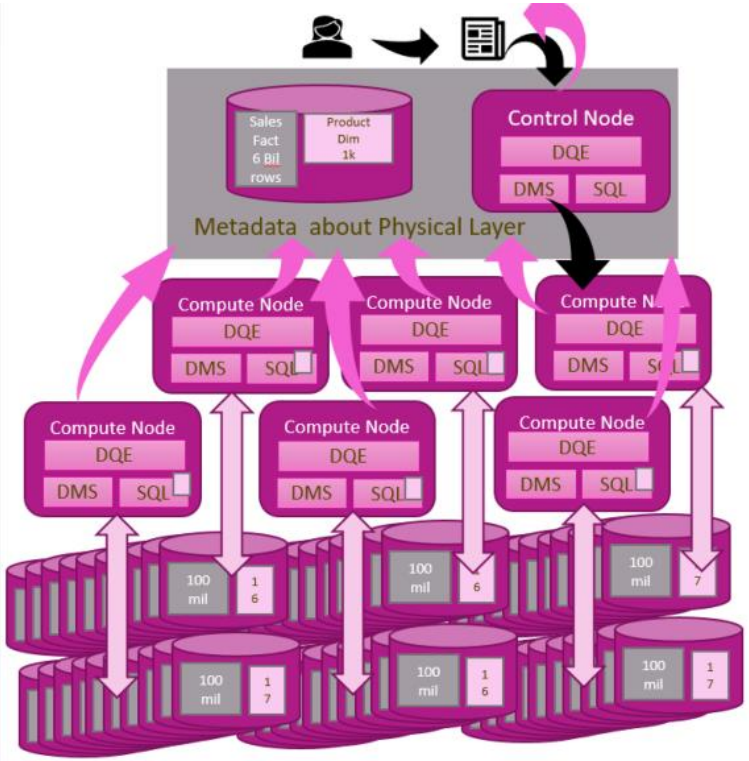
Synapse Studio is a unifying experience to bring all aspects of the modern data warehouse in to one development environment.

And simplify leveraging scalable compute and querying across Data Lake storage and the relational DB.

This presentation focuses on SQL DB.

DB MPP Architecture

- Logical layer
  No persistent data stored
- Distributed query engine
- Data movement
- Scalable compute
- Physical layer
  60 Distributions

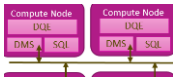


User submits a SQL query referencing tables as known at the Logical Layer.

Using meta data about how data is stored at physical layer, control node creates execution plan and sends plan to each compute node.

One arrow shown, should be 6 in diagram

Execution plan is a mix of SQL and Data Movement (DMS). Each Compute Node works with a specific portion of the 60 distributions at the physical layer. If all data required for the query is not present on a Compute Node's set of data Distributions then DMS moves copies of data as needed.



Based on MPP plan DMS uses SQL to pull data, transfers to other DMS where SQL writes data

Once data is present to answer query, each compute node executes SQL on their data

and passes back results to Control node.

Control node does any final aggregations and returns results to user.

MPP Execution

We learned that MPP execution requires:

1. Control Node to determine how logical layer (queried) maps to physical layer
2. Control Node must determine what data needs to be moved to resolve the query
3. Control Node must determine the Plan to split up the work across Compute Nodes and then combine the results
4. Control Node sends the Plan to Compute Nodes
5. Compute Nodes through DMS and SQL move data temporarily across 60 Distributions
6. Compute Nodes SQL Server creates SQL Execution plan for each query to run
7. Executes and Returns Results to Control Node
8. Control Node does any final aggregations of results and returns to client

That is a lot of overhead, but if you are scanning and joining very large tables to perform some analytic queries it is worth it.

It is not worth it if you are looking up a single record, inserting or updating a single record.

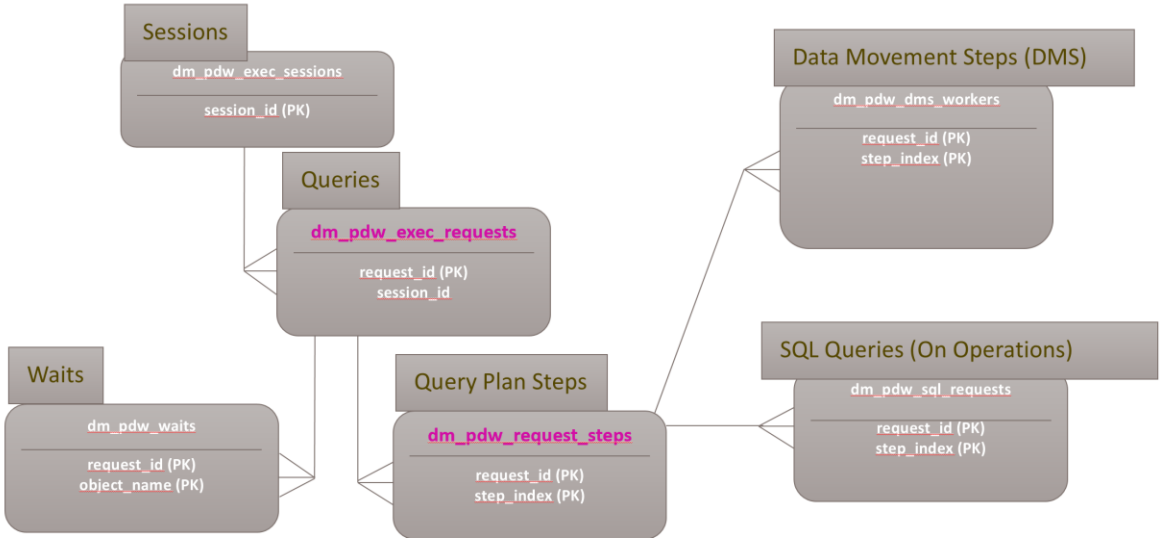
Synapse DB is for scan centric analytic workloads.

MPP Execution

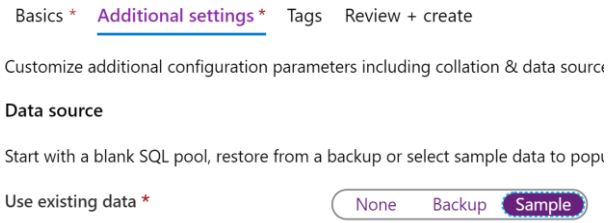
The right workload (SCAN CENTRIC) matters, but we can impact how much work the MPP plan has to do by being aware of:

- Fundamentals: Table Distribution Types and Storage
- Distribution Keys and Skew
- Coding Best Practices

We can also look and see how much impact we are having by using built in Data Management Views (DMVS).



DEMO1 – impact of default table and storage types



When creating Azure Synapse Analytics, go to 'Additional Settings' and select 'Sample'.

Script to the right creates 3 tables using SELECT INTO which creates tables with defaults of ROUND_ROBIN and COLUMN CLUSTERED INDEX.

Then creates 3 tables using CREATE TABLE AS SELECT specifying the DISTRIBUTION and STORAGE.

```
---USING SYNAPSE DB - when created Synapse DB selected SAMPLE so got Adventure Works

select count(*) from salesbyregion           --4,397,454
select TOP 5 *  from salesbyregion
--SalesAmountPostalCodeStateProvinceCode
--53.992300NSW
--2181.56252300NSW
--1000.437553131NW
--49.9989502NV
--2443.352300NSW
select distinct postalcode from salesbyregion  -- 635

select count(*) from [dbo].[DimGeography]  --655

SELECT * INTO DG_Def_Def FROM [DimGeography]   --DEFAULT DISTRIBUTION and STORAGE
CREATE TABLE DG_Rep_Heap WITH (DISTRIBUTION = REPLICATE, HEAP)
AS select * from [DimGeography]


SELECT * INTO SBR_Def_Def FROM salesbyregion   --DEFAULT DISTRIBUTION and STORAGE
SELECT * INTO SBR2_Def_Def FROM salesbyregion  --DEFAULT DISTRIBUTION and STORAGE

CREATE TABLE SBR_Dist_CI WITH (DISTRIBUTION = HASH(postalcode), CLUSTERED
INDEX(SalesAmount))
AS select * from salesbyregion

CREATE TABLE SBR2_Dist_CI WITH (DISTRIBUTION = HASH(postalcode), CLUSTERED
INDEX(SalesAmount))
AS select * from salesbyregion
```

DEMO1 – identical SELECT statements against Default and User Defined Table Types

```
SELECT D.City,SUM(F1.SalesAmount),Count(F2.PostalCode)
FROM SBR_Def_Def F1
JOIN SBR2_Def_Def F2 on F1.PostalCode = F2.PostalCode
JOIN DG_Def_Def D on D.PostalCode = F1.PostalCode
GROUP BY D.City
OPTION (LABEL = 'Query1 Table Defaults')
```

Default

```
SELECT D.City,SUM(F1.SalesAmount),Count(F2.PostalCode)
FROM SBR_Dist_CI F1
JOIN SBR2_Dist_CI F2 on F1.PostalCode = F2.PostalCode
JOIN DG_Rep_Heap D on D.PostalCode = F1.PostalCode
GROUP BY D.City
OPTION (LABEL = 'Query1 Table Specified')
```

User Defined

Execute twice, eliminate caching impact on query times, and force BuildReplicatedTableCache for new Replicate table

DEMO1 – Pull the Results

```
--SEE ALL RESULTS
 SELECT * FROM sys.dm_pdw_exec_requests R
 WHERE R.request_id  IN ----GET MOST RECENT EXECUIONS OF ABOVE QUERIES
 (Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] = 'Query1 Table Defaults'
 UNION
 Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] = 'Query1 Table Specified')

  SELECT R.[Label],S.* FROM sys.dm_pdw_request_Steps S
             JOIN sys.dm_pdw_exec_requests R on S.request_id = R.request_id
  WHERE S.request_id  IN ----GET MOST RECENT EXECUIONS OF ABOVE QUERIES
 (Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] = 'Query1 Table Defaults'
 UNION
 Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] = 'Query1 Table Specified')
 ORDER BY 1,3

--SEE KEY RESULT COLUMNS
  SELECT R.[Label],S.step_index,operation_type,location_type,s.total_elapsed_time,row_count FROM
sys.dm_pdw_request_Steps S
             JOIN sys.dm_pdw_exec_requests R on S.request_id = R.request_id
  WHERE S.request_id  IN ----GET MOST RECENT EXECUIONS OF ABOVE QUERIES
 (Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] = 'Query1 Table Defaults'
 UNION
 Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] = 'Query1 Table Specified')
 ORDER BY 1,2
```

DEMO1 – Results

Label	step_index	operation_type	location_type	total_elapsed_time	row_count
Query1 Table Defaults	0	RandomIDOperation	Control	0	-1
Query1 Table Defaults	1	OnOperation	Compute	156	-1
Query1 Table Defaults	2	ShuffleMoveOperation	Compute	1797	4397454
Query1 Table Defaults	3	RandomIDOperation	Control	0	-1
Query1 Table Defaults	4	OnOperation	Compute	187	-1
Query1 Table Defaults	5	ShuffleMoveOperation	Compute	1734	4397454
Query1 Table Defaults	6	RandomIDOperation	Control	0	-1
Query1 Table Defaults	7	OnOperation	Compute	156	-1
Query1 Table Defaults	8	ShuffleMoveOperation	Compute	500	655
Query1 Table Defaults	9	RandomIDOperation	Control	0	-1
Query1 Table Defaults	10	OnOperation	Compute	171	-1
Query1 Table Defaults	11	ShuffleMoveOperation	Compute	4078	646
Query1 Table Defaults	12	ReturnOperation	Compute	156	-1
Query1 Table Defaults	13	OnOperation	Compute	109	-1
Query1 Table Defaults	14	OnOperation	Compute	156	-1
Query1 Table Defaults	15	OnOperation	Compute	156	-1
Query1 Table Defaults	16	OnOperation	Compute	140	-1

Default

Label	step_index	operation_type	location_type	total_elapsed_time	row_count
Query1 Table Specified	0	RandomIDOperation	Control	0	-1
Query1 Table Specified	1	OnOperation	Compute	171	-1
Query1 Table Specified	2	ShuffleMoveOperation	Compute	1734	646
Query1 Table Specified	3	ReturnOperation	Compute	125	-1
Query1 Table Specified	4	OnOperation	Compute	125	-1

User Defined

Agenda

- Azure SQL DW to Synapse Analytics
- DB Massive Parallel Processing (MPP) Architecture
- MPP Execution
- **Fundamentals: Table Distribution Types and Storage**
- **Distribution Keys and Skew**
- **Coding Best Practices**
- Questions

Table DISTRIBUTION and STORAGE TYPE

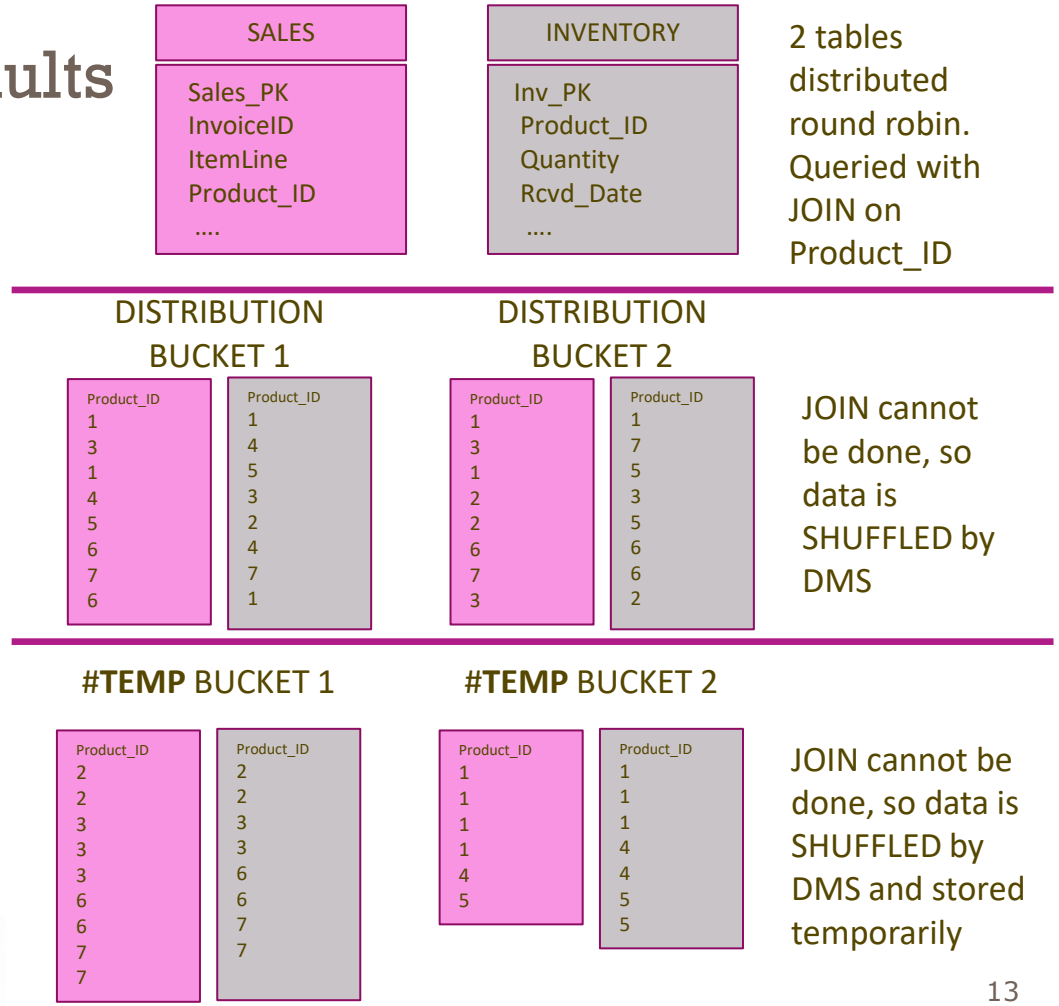
- HASH
  distributed across 60 buckets based on byte pattern, all like values go to same bucket
- **ROUND_ROBIN**
  distributed randomly and evenly across 60 buckets
- REPLICATE
  distributed ROUND_ROBIN but on first query full copy to each COMPUTE NODE (repeated whenever changed)
- **COLUMN CLUSTERED INDEX**
  table data stored as COLUMN COMPRESSED. Up to 15x compression. Fast query on non-strings.
- CLUSTERED INDEX
  Row storage, rows physically stored ordered by CI Column(s). Can help limit range scans.
- HEAP
  Row storage, no ordering. Fast writes. Good for small tables.

Why not always use defaults

- **ROUND_ROBIN**

  This is fine if your large table is only ever joined to small replicate tables.

  If it is ever joined to HASH table, or another ROUND_ROBIN table must move to satisfy the join.



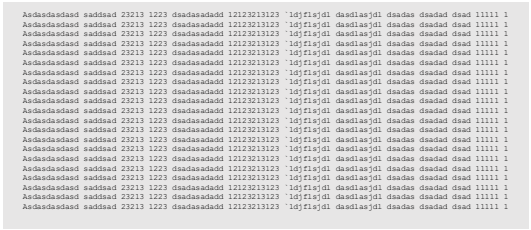
Why not always use defaults

- **COLUMN CLUSTERED INDEX** table data stored as COLUMN COMPRESSED. Up to 15x compression. Fast query on non-strings.

String values in WHERE clause need uncompressed Column Store to evaluate. Numeric in WHERE uses metadata in Column Store pages to determine if any values meet criteria.

CCI is not always best choice, watch for STRING criteria small tables.

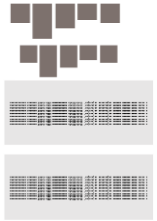
Row Group



Data comes in as rows and is compressed in batches 1 million

Column Store

Compressed data smaller space, faster reads.



CCI end up being a mix of Column Compressed and row storage.
A billion compressed rows and 900 thousand row still beneficial.

Small table under 60 million (60 buckets) never autocompressed, needs ALTER INDEX executed.

Choosing a Distribution Key

- Three Basic Rules
  - Avoid Data Skew
  - Minimize Data Movement (JOINS , GROUP BY)
  - Provide Balanced Execution

- Choose a column that
  - Has a high number of distinct values
  - Is defined as NOT NULL
  - Doesn’t contain dominating values

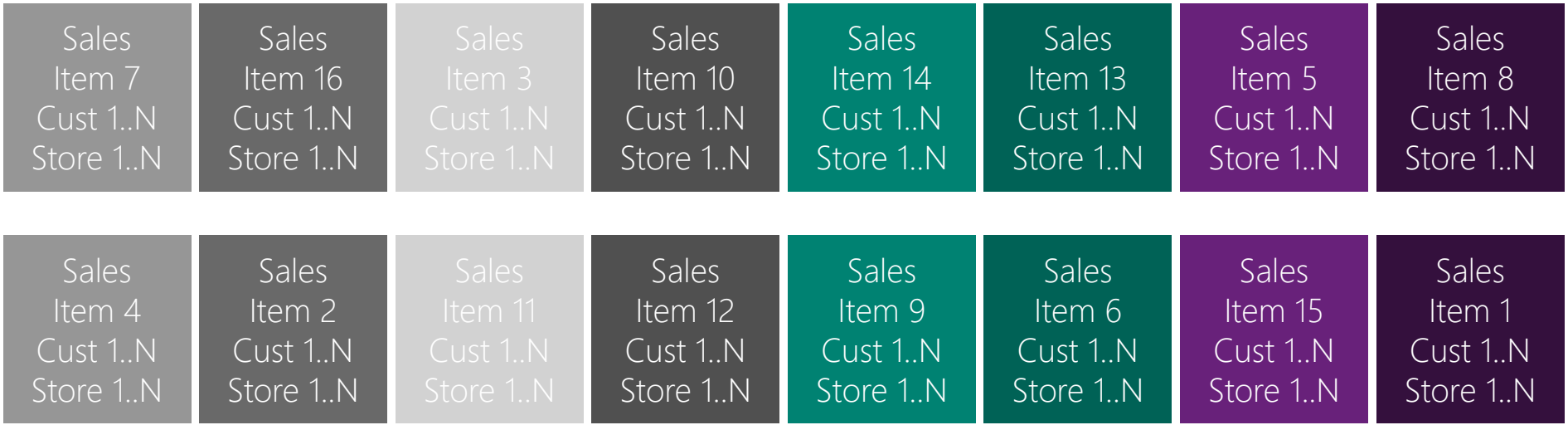
Minimize Data Movement

- More important than skew
- Movement can re-introduce skew
- Pay close attention to:
  - Distributed tables that self-join
  - Distributed tables that join to each other
  - Outer joins – check for compatibility or convert
- Where possible:
  - Ensure the join contains the distribution key

Sales Table – Distributed by Item

What are the top 10 bestselling items?

Which is the top selling store?



How many sales of item 3 did we have?

Sales Table – Distributed by Store

What are the top 10 bestselling items?

Which is the top selling store?

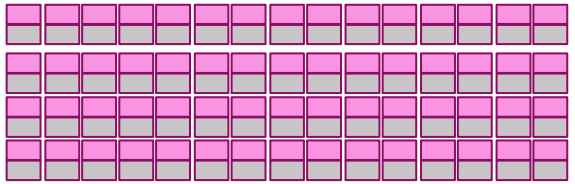
There are 60 Distributions.
Consider that most chains have far more items than they do stores.

Minimize Data Movement

SALES
*Sales_PK InvoiceID ItemLine Product_ID ….

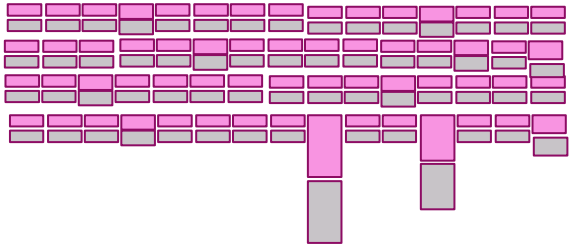
INVENTORY
*Inv_PK Product_ID Quantity Rcvd_Date ….

Very little skew, even spread across 60 distributions.



```
SELECT ….
FROM SALES S JOIN INVENTORY I
ON S.Product_ID = I.Product_ID
WHERE …..
```

Both tables shuffled to answer this query.



1. If frequent join on Product_ID then better to just DISTRIBUTE table regardless of skew.

2. If workload is mainly queries of individual tables (SALES,INVENTORY) joined to REPLICATE then top.

3. Even if 2 is true, if 100 billion rows maybe distribute with skew to avoid shuffle (as 2 has no shuffle either way)

Coding Best Practices

- Never use “SELECT * INTO”
  always use CREATE TABLE so you control table distribution and storage.

- Never CREATE TABLE and then do an INSERT
  always use (CTAS) CREATE TABLE AS SELECT non logged operation and runs fully parallelized.

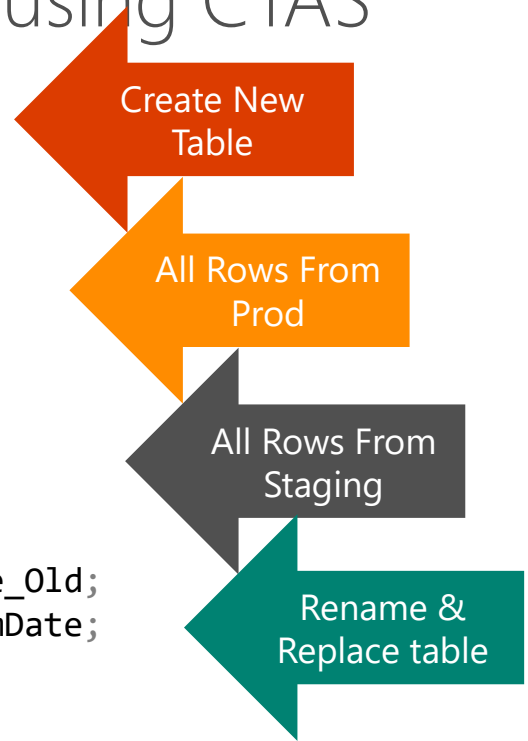
- Avoid directly doing large volumes of INSERT, UPDATE,DELETE
  consider using CTAS

CTAS – Small Table "Insert"

Recreate the table using CTAS

```
CREATE TABLE dbo.DimDate_New
WITH (DISTRIBUTION = REPLICATE
, CLUSTERED INDEX (DateKey ASC)
)
AS
SELECT *
FROM   dbo.DimDate  AS prod
UNION ALL
SELECT *
FROM   dbo.DimDate_stg AS stg
;

RENAME OBJECT DimDate TO DimDate_Old;
RENAME OBJECT DimDate_New TO DimDate;
```



CTAS Upsert

```
CREATE TABLE dbo.DimProduct_upsert
WITH (Distribution=HASH(ProductKey)
, CLUSTERED INDEX (ProductKey)
)
AS -- Update
SELECT      s.ProductKey
,           s.EnglishProductName
,           s.Color
FROM        dbo.DimProduct p
JOIN        dbo.stg_DimProduct s ON p.ProductKey = s.ProductKey
UNION ALL--Keep rows that do not exist in stage
SELECT      p.ProductKey
,           p.EnglishProductName
,           p.Color
FROM        dbo.DimProduct p
LEFT JOIN   dbo.stg_DimProduct s ON p.ProductKey = s.ProductKey
WHERE       s.ProductKey IS NULL
UNION ALL--Inserts
SELECT      s.ProductKey
,           s.EnglishProductName
,           s.Color
FROM        dbo.DimProduct p
RIGHT JOIN dbo.stg_DimProduct s ON p.ProductKey = s.ProductKey
WHERE       p.ProductKey IS NULL
```

Performance Optimisation
Code is less maintainable

DEMO2 – ROUND_ROBIN single table query
Try these at home (or in the cloud rather)
depends on executing code from demo 1

```
-------ROUND_ROBIN fine against one table
--1A
 SELECT SUM(F1.SalesAmount)
 FROM SBR_Def_Def F1
 OPTION (LABEL = 'Query2 Single Table Defaults')

  SELECT SUM(F1.SalesAmount)
 FROM SBR_Dist_CI F1
 OPTION (LABEL = 'Query2 Single Table Specified')

--1B
 --SEE KEY RESULT COLUMNS
  SELECT R.[Label],S.step_index,operation_type,location_type,s.total_elapsed_time,row_count FROM sys.dm_pdw_request_Steps S
            JOIN sys.dm_pdw_exec_requests R on S.request_id = R.request_id
  WHERE S.request_id  IN ----GET MOST RECENT EXECUIONS OF ABOVE QUERIES
 (Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] = 'Query2 Single Table Defaults'
 UNION
 Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] =  'Query2 Single Table Specified')
 ORDER BY 1,2
```

ROUND_ROBIN same number steps as HASH table, when single table and no group by

	Label	step_index	operation_type	location_type	total_elapsed_time	row_count
1	Query2 Single Table Defaults	0	OnOperation	Control	31	-1
2	Query2 Single Table Defaults	1	PartitionMoveOperation	Compute	437	60
3	Query2 Single Table Defaults	2	ReturnOperation	Control	0	-1
4	Query2 Single Table Defaults	3	OnOperation	Control	0	-1
5	Query2 Single Table Specified	0	OnOperation	Control	31	-1
6	Query2 Single Table Specified	1	PartitionMoveOperation	Compute	2687	60
7	Query2 Single Table Specified	2	ReturnOperation	Control	15	-1
8	Query2 Single Table Specified	3	OnOperation	Control	0	-1

DEMO2 – look at CCI compressed rows

```
 --2A
 SELECT t.name AS TableObjectName
,SUM(rg.total_rows - rg.deleted_rows) AS DELTAStoreRowCount
,SUM(CASE
WHEN rg.state_desc = 'OPEN' THEN rg.total_rows - rg.deleted_rows
ELSE 0
END) AS DELTAStoreOpenCount
,SUM(CASE
WHEN rg.state_desc = 'CLOSED' THEN rg.total_rows - rg.deleted_rows
ELSE 0
END) AS DELTAStoreClosedCount
,SUM(CASE
WHEN rg.state_desc = 'COMPRESSED' THEN rg.total_rows - rg.deleted_rows
ELSE 0
END) AS DELTAStoreCompressedCount
,MIN(CASE WHEN rg.state_desc = 'OPEN' THEN rg.created_time ELSE NULL END)  DELTAStoreMinOpenedDate
,MAX(CASE WHEN rg.state_desc = 'COMPRESSED' THEN rg.created_time ELSE NULL END)  DELTAStoreMaxCompressedDate
FROM sys.tables t
JOIN sys.indexes i                              ON  t.object_id      = i.object_id
JOIN sys.pdw_index_mappings im                  ON  i.object_id      = im.object_id
AND i.index_id       = im.index_id
JOIN sys.pdw_nodes_indexes ni                   ON  im.physical_name = ni.name
AND im.index_id      = ni.index_id
JOIN sys.dm_pdw_nodes_db_column_store_row_group_physical_stats rg ON  ni.object_id     = rg.object_id
AND ni.distribution_id = rg.distribution_id
WHERE t.name in ('SBR_Def_Def','SBR2_Def_Def','DG_Def_Def')
group by t.name
```

After compressing on next slides tables look like this:

TableObjectName	DELTAStoreRowCount	DELTAStoreOpenCount	DELTAStoreClosedCount	DELTAStoreCompressedCount	DELTAStoreMinOpenedDate	DELTAStoreMaxCompressedDate
DG_Def_Def	655	0	0	655	NULL	2020-04-12 19:16:42.873
SBR_Def_Def	4397454	0	0	4397454	NULL	2020-04-12 19:16:37.737
SBR2_Def_Def	4397454	0	0	4397454	NULL	2020-04-12 19:16:41.303

DEMO2 – look at CCI compressed rows –Part 2

```
----------------------------------------
--2B  --compress
 ALTER INDEX ALL ON dbo.SBR_Def_Def REBUILD;
 ALTER INDEX ALL ON dbo.SBR2_Def_Def REBUILD;
 ALTER INDEX ALL ON dbo.DG_Def_Def REBUILD;
 --rerun 2a

 --3A  lets execute against compressed
 SELECT D.City,SUM(F1.SalesAmount),Count(F2.PostalCode)
 FROM SBR_Def_Def F1
 JOIN SBR2_Def_Def F2 on F1.PostalCode = F2.PostalCode
 JOIN DG_Def_Def D on D.PostalCode = F1.PostalCode
 GROUP BY D.City
 OPTION (LABEL = 'Query1 Table Defaults Compressed')

--3B
  SELECT * FROM sys.dm_pdw_exec_requests R
 WHERE R.request_id  IN ----GET MOST RECENT EXECUIONS OF ABOVE QUERIES
 (Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] = 'Query1 Table Defaults'
 UNION
 Select max(request_id) request_id from sys.dm_pdw_exec_requests where [Label] = 'Query1 Table Defaults Compressed')
```

total_elapsed_time	label	e
16126	Query1 Table Defaults	N
8797	Query1 Table Defaults Compressed	N

Significant performance boost once compressed

DEMO2 – look at SKEW

```
--3C look at skew
--remember we distributed by postal_code lets see how even the spread
select p.row_count,nt.name
FROM   sys.tables t
JOIN sys.pdw_table_mappings m ON t.object_id = m.object_id
    JOIN sys.pdw_nodes_tables nt ON nt.name = m.physical_name
JOIN [sys].[dm_pdw_nodes_db_partition_stats] pON p.object_id = nt.object_id
AND p.pdw_node_id = nt.pdw_node_id
AND p.distribution_id = nt.distribution_id
where t.name = 'SBR_Dist_CI'
order by 1
```

	row_count	name
1	25817	Table_b6c1c034b0304f0998079387428d0f15_29
2	33507	Table_b6c1c034b0304f0998079387428d0f15_46
3	34577	Table_b6c1c034b0304f0998079387428d0f15_57
4	41683	Table_b6c1c034b0304f0998079387428d0f15_54
5	42209	Table_b6c1c034b0304f0998079387428d0f15_45
6	44540	Table_b6c1c034b0304f0998079387428d0f15_15
7	44996	Table_b6c1c034b0304f0998079387428d0f15_5
8	47461	Table_b6c1c034b0304f0998079387428d0f15_1
9	47661	Table_b6c1c034b0304f0998079387428d0f15_30
10	47673	Table_b6c1c034b0304f0998079387428d0f15_52
11	48150	Table_b6c1c034b0304f0998079387428d0f15_27
12	48354	Table_b6c1c034b0304f0998079387428d0f15_42

50	101606	Table_b6c1c034b0304f0998079387428d0f15_60
51	106200	Table_b6c1c034b0304f0998079387428d0f15_16
52	106475	Table_b6c1c034b0304f0998079387428d0f15_35
53	106513	Table_b6c1c034b0304f0998079387428d0f15_59
54	109519	Table_b6c1c034b0304f0998079387428d0f15_39
55	110697	Table_b6c1c034b0304f0998079387428d0f15_31
56	111407	Table_b6c1c034b0304f0998079387428d0f15_21
57	117417	Table_b6c1c034b0304f0998079387428d0f15_43
58	121087	Table_b6c1c034b0304f0998079387428d0f15_34
59	125291	Table_b6c1c034b0304f0998079387428d0f15_44
60	127226	Table_b6c1c034b0304f0998079387428d0f15_9

Not very even distribution, smallest bucket has 25k largest 127k.

1 query would be 5 times longer than the fastest. Probably doesn't matter on 4 million rows, but big impact if this much skew on billions of rows.

BuildReplicatedTableCache - Expensive

MSFT recommends Replicate table if under 2GB in size.

I used a table approximately .75 GB.

seconds	command
206	BuildReplicatedTableCache [Adv_Works].[dbo].[Test2_CI]
171	BuildReplicatedTableCache [Adv_Works].[dbo].[Test2_HEAP]
136	BuildReplicatedTableCache [Adv_Works].[dbo].[Test2_CCI]

Then tested with a 3GB table.

seconds	command
945	BuildReplicatedTableCache [Adv_Works].[dbo].[Test2_CI]
886	BuildReplicatedTableCache [Adv_Works].[dbo].[Test2_HEAP]
800	BuildReplicatedTableCache [Adv_Works].[dbo].[Test2_CCI]

When ran two in parallel (meaning query with 2 table references after a change) saw a 40% increase in time for the BuildReplicateTableCache. Higher DWCU did not impact build time.

select H.ID, H.Request_ID, count(C.request_id) from Test2_Base H join Test2_CCI C on H.Request_ID = C.Request_id group by H.ID, H.Request_ID

During BRTC – 21 seconds first time

step_index	operation_type	milliseconds
0	RandomIDOperation	0
1	OnOperation	109
2	ShuffleMoveOperation	1875
3	RandomIDOperation	0
4	OnOperation	109
5	ShuffleMoveOperation	2453
6	ReturnOperation	7984
7	OnOperation	125
8	OnOperation	124

After BRTC 6.8 Seconds

step_index	operation_type	milliseconds
0	ReturnOperation	6812

SHUFFLE not so pretty either.