SQL Data Warehouse
Databases and Tables

Chris Testa-O'Neill
Features Engineer
Analytics and Data Science Team

Microsoft

Agenda

Databases

Tables

Indexes

Statistics

Partitions

Databases

CREATE DATABASE Command

```
CREATE DATABASE database_name [ COLLATE collation_name ]
(
[ MAXSIZE = { 250 | 500 | 750 | 1024 | 5120 | 10240 | 20480
| 30720 | 40960 | 51200 | 61440 | 71680 | 81920 | 92160 |
102400 | 153600 | 204800 | 245760 } GB ,]


EDITION = 'datawarehouse',


SERVICE_OBJECTIVE = { 'DW100' | 'DW200' | 'DW300' | 'DW400'
| 'DW500' | 'DW600' | 'DW1000' | 'DW1200' | 'DW1500' |
'DW2000' | 'DW3000' | 'DW6000' }
) [;]
```

ALTER DATABASE Command

```
ALTER DATABASE database_name
MODIFY NAME = new_database_name |
MODIFY ( <edition_option> [, ... n] )


<edition_option> ::= MAXSIZE = { 250 | 500 | 750 | 1024 |
5120 | 10240 | 20480 | 30720 | 40960 | 51200 | 61440 |
71680 | 81920 | 92160 | 102400 | 153600 | 204800 | 245760 }
GB |


SERVICE_OBJECTIVE = { 'DW100' | 'DW200' | 'DW300' | 'DW400'
| 'DW500' | 'DW600' | 'DW1000' | 'DW1200' | 'DW1500' |
'DW2000' | 'DW3000' | 'DW6000'}
```

CREATE SCHEMA Command

```
CREATE SCHEMA schema_name [ AUTHORIZATION owner_name ] [;]
```

schema_name

- Is the name by which the schema is identified within the database.

AUTHORIZATION owner_name

- Specifies the name of the database-level principal that will own the schema. This principal may own other schemas, and may not use the current schema as its default schema.

Demo:
Creating a Database

Tables

Why Distribute Data

- Divide & conquer: lots of small queries to solve
- Evenly spreading the data leads to even use of the appliance resources

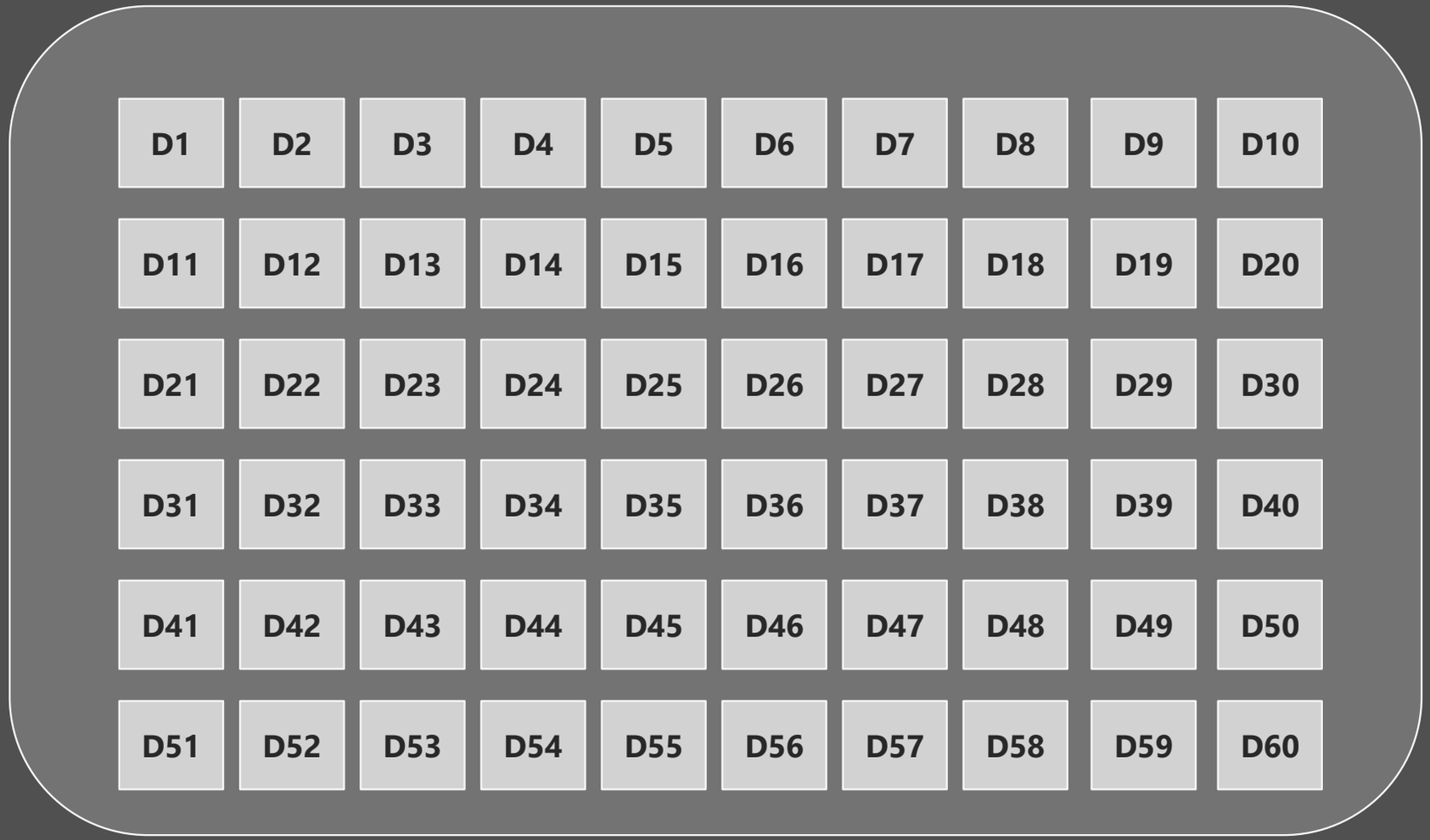
SQL Table Geometries

Distributed:

- A table structure that is distributed across all MPP nodes of the Data Warehouse Database
  - HASH: where the value of a single column gets hashed to define the distribution number where the records will get inserted
  - ROUND_ROBIN: where the records are distributed in a “round robin” manner across all distributions

Distributions

```
CREATE TABLE myTable (column Defs)
WITH ( DISTRIBUTION = ROUND_ROBIN | HASH (id));
```



To HASH or ROUND_ROBIN?

Use HASH distributed tables when:

- For fact/detail tables if you **DO** have a common distribution key which **is used** in multi-distributed table joins
- If Full Table Scans do not provide acceptable performance

Use ROUND_ROBIN distributed tables when:

- On "Azure SQL Data Warehouse": instead of REPLICATED tables
- Multiple "primary/foreign key"-like joins are common
- When the data is an initial loading table

(Remember: Data movement will be most likely be involved in "multi distributed table joins", if at least one of them is ROUND_ROBIN distributed)

CREATE TABLE Command

```
CREATE TABLE [database_name.][schema_name.]table_name
   (
     column_name <data_type>
          [ COLLATE Windows_collation_name ]
          [ NULL | NOT NULL ] }
          [ [ CONSTRAINT constraint_name ]
                  DEFAULT constant_expression ] [ ,...n ]
   )
[ WITH ( <table_option> [ ,...n ] ) ]
[;]
```

CREATE TABLE Command

Temporary tables

```
CREATE TABLE [database_name.][schema_name.]#table_name
   (
     column_name <data_type>
          [ COLLATE Windows_collation_name ]
          [ NULL | NOT NULL ] }
          [ ,...n ]
   )
WITH ( LOCATION = USER_DB [, <table_option> [ ,...n ] ] )
[;]
```

CREATE TABLE Command (continued)

```
<table_option> ::=
    CLUSTERED COLUMNSTORE INDEX
    | CLUSTERED INDEX
       ( { index_column_name [ ASC | DESC ] } [ ,...n ] )
    | DISTRIBUTION = {
                       HASH ( distribution_column_name )
                     | ROUND_ROBIN
                     }
    | PARTITION
        ( partition_column_name RANGE [ LEFT | RIGHT ]
          FOR VALUES ( [ boundary_value [,...n] ] )
        )
```

Create Table Command - Example

```
CREATE TABLE myTable
        (
            id integer NOT NULL,
            lastName varchar(20),
            zipCode varchar(6)
        );
```

When distribution option is not specified, defaults to ROUND_ROBIN

Creating Tables - Limitations

Table limitations

- 2 billion tables per database
- Up to 1,024 columns per table
- Number of rows limited only by available storage
- Maximum bytes per row is 8,060

Current Limitations of Data Types

Most Scalar data types supported by SQL Server are supported by the MPP DWH

Main exceptions

- Text (and related BLOB data types, including `xxxxxx(MAX)`)
- XML
- SQL variant
- Timestamp
- System and CLR UDTs

IDENTITY columns and PK/FK constraints not supported

Collations are fully supported, similar to SQL Server, on the Column Level

- Default: SQL_Latin1_General_CI_AS_KS_WS
  also possible: Latin1_General_100_CI_AS_KS_WS (on the DB level)

SQL Server Data Types	PDW
bigint	✓
binary	✓
bit	✓
char/nchar	✓
date, time	✓
datetime	✓
datetime2	✓
datetimeoffset	✓
decimal	✓
float	✓
geography / geometry	
hierarchyid	
image	
int	✓
money	✓
numeric	
real	✓
smalldatetime	✓
smallint	✓
smallmoney	✓
sql_variant	
sysname	
text / ntext	
timestamp	
tinyint	✓
uniqueidentifier	
varbinary	✓
varchar / nvarchar	✓
xml	

Constraints

Primary keys and unique indexes are *not* supported

Foreign constraints are *not* supported

Default constraints are supported

- CREATE TABLE
- ALTER TABLE
- Must be a literal value or a constant

ALTER TABLE Command

Modify/Add/Drop Column

Cannot be modified or dropped

- Distribution column
- Column included in index
- Partition column

REBUILD (all or specified partitions)

SPLIT/MERGE/SWITCH Partition

ALTER TABLE

```
ALTER TABLE [ database_name . [schema_name ] . | schema_name. ] source_table_name
{
    ALTER COLUMN column_name
        {
            type_name [ ( precision [ , scale ] ) ]
            [ COLLATE Windows_collation_name ]
            [ NULL | NOT NULL ]
        }
    | ADD { <column_definition> | <column_constraint> FOR column_name} [ ,...n ]
    | DROP { COLUMN column_name | [CONSTRAINT] constraint_name } [ ,...n ]
    | REBUILD [ PARTITION = { ALL | partition_number } ]
    | { SPLIT | MERGE } RANGE (boundary_value)
    | SWITCH [ PARTITION source_partition_number
      TO target_table_name [ PARTITION target_partition_number ]
}
[;]
```

ALTER TABLE

```
<column_definition>::=
{
    column_name
    type_name [ ( precision [ , scale ] ) ]
    [ <column_constraint> ]
    [ COLLATE Windows_collation_name ]
    [ NULL | NOT NULL ]
}

<column_constraint>::=
    [ CONSTRAINT constraint_name ] DEFAULT constant_expression
```

TRUNCATE TABLE Command

Works with distributed tables

Works with permanent or temporary tables

Not allowed

- with EXPLAIN
- within user-defined transaction

Performance

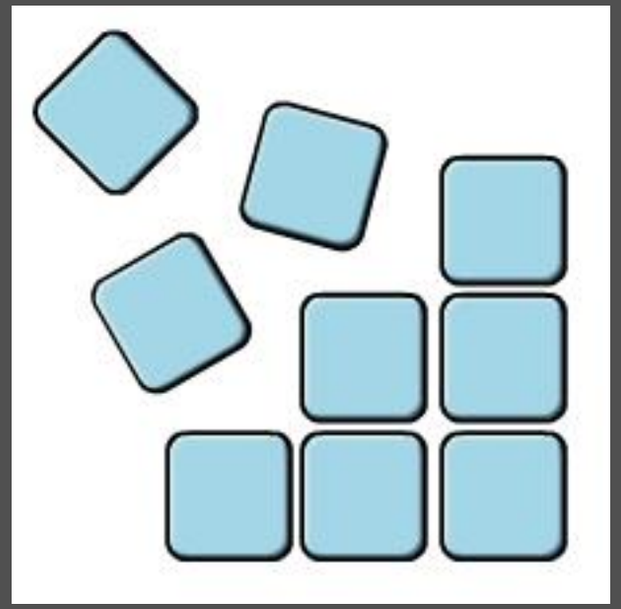
- For a replicated table, TRUNCATE TABLE is executed in parallel across the compute nodes
- For a distributed table, TRUNCATE TABLE is executed in parallel across the compute nodes, and serially across the distributions within each compute node

Demo:
Creating Tables

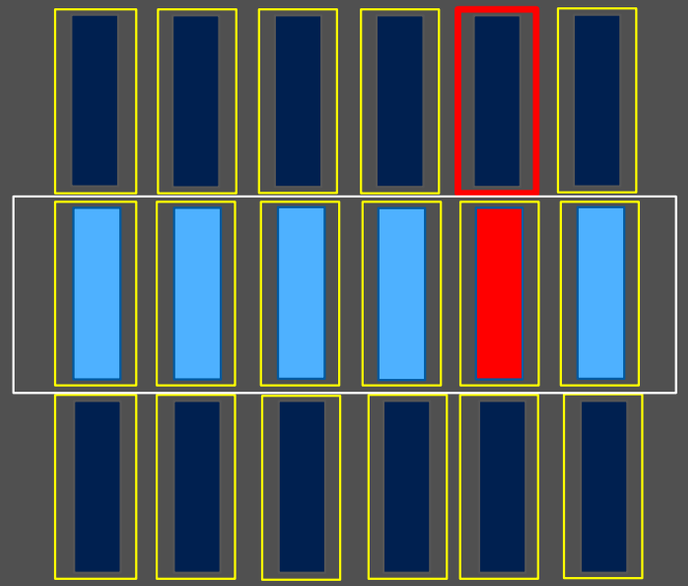
Indexes

Indexing options

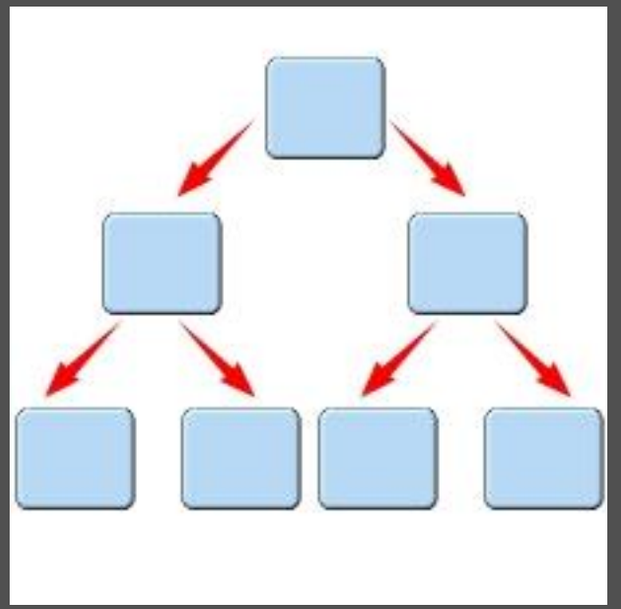
Heap



Columnstore



Clustered Index



Clustered Columnstore Overview

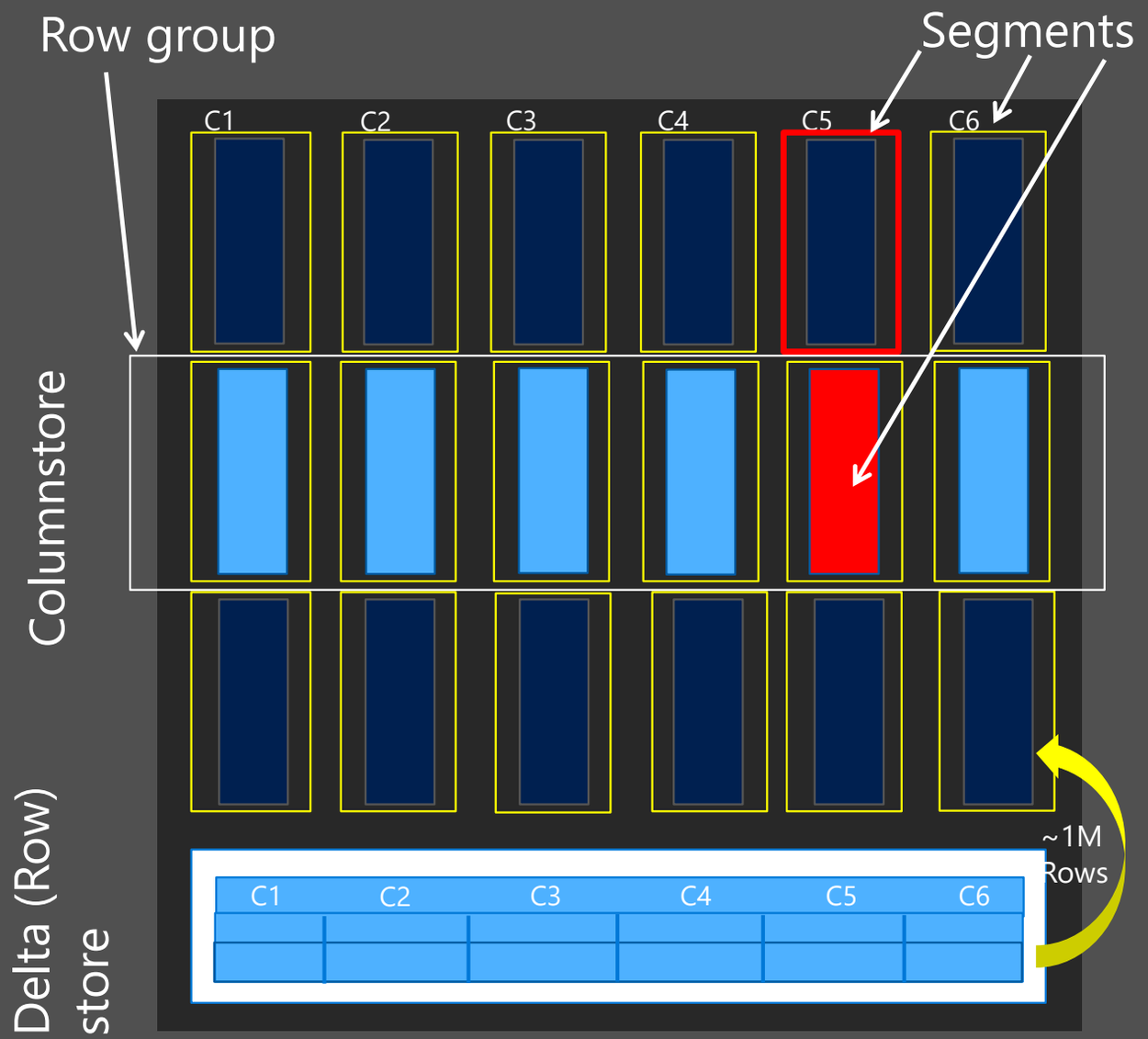
Introduction

- Clustered Columnstore indexes are designed for data warehouse type queries where only a portion of the table columns are required.
- Enables users to isolate the required data far more efficiently than with traditional row-based storage.
- Typically provides higher compression ratios due to tables generally containing more duplicate values in a column than a row:
  - Page compression is normally ~ 2.5x to 3.5x depending on data.
  - Columnstore is normally ~ 5x to 15x depending on data.
- Higher compression ratios contribute to a greater ROI for raw storage.
- New batch mode processing enables lower CPU utilization for the same number of rows processed.
- Supports important existing data warehouse functionality such as partition switching, splitting, and merging.

Clustered Column Store Index Design

How DML is supported

- Changes to data can be applied directly to a clustered columnstore index
- INSERTs are added to the deltastore table
  - NOTE: The deltastore table is a **Page Compressed Heap**
- DELETEs from columnstore are logical; data is not physically removed until **REBUILD** is issued
  - DELETE from deltastore is physical as it is row based
- UPDATE is INSERT + DELETE
- Deltastore is automatically converted to columnStore at ~1M rows by background "Tuple Mover" process
  - Can also be forced with **REORGANIZE** at ~1M rows
- Converting deltastore to columnstore with **REORGANIZE** is an **ONLINE** operation
- **ADD** / **DROP** / **ALTER COLUMN** is supported
  - Partition switching also supported



Clustered Index vs. CCI vs. Heap Table

Clustered Index (CI)

- Introduces a sort step in table loads (loads slower than heap)
- Requires periodic maintenance to defragment (CTAS)
- Cluster fact tables on same column used for partitioning for optimal sequential I/O

Clustered Columnstore Index (CCI)

- Offers highest compression
- Optimized for Data Warehouse House (DWH) Workload

Heap Table

- Have fastest rate for inserts/loads ➔ e.g. for “store only” tables

Additional Considerations

- Minimize secondary (non-clustered) indexes with MPP DWH to reduce random I/O

Non-Clustered Indexes

Use “judiciously”

- Will become fragmented with DML
    - Use Alter Index to defrag
- Will affect performance of DML operations
- Will take disk space

Statistics

What are Statistics?

Database objects!

- Contain statistical information about the distribution of values in a column
  - The statistics object includes a histogram to show the distribution of values in the first column
- Multi-column statistics can also be generated
  - These also hold density information i.e. the correlation of values between columns

Statistics on Compute and Control nodes

Statistics on compute nodes

- The MPP Engine uses automatic statistics settings on compute nodes
- Compute node statistics use standard SQL Server sampling
    - Updated after table changes by 20 percent
- Can use compound (covering) statistics

Statistics on control node

- The MPP Engine uses statistics on control node to improve performance by using cardinal estimation
- Do **not** updated automatically!

Multicolumn statistics

If your joins are on multiple key columns, multicolumn stats help avoid mistaken “nested loop” plans on SQL Server node

Example:

- A long-running query step involved multicolumn join key and was experiencing lots of reads
- SQL query plan (estimated) showed nested loop and a very small number of estimated result rows from the join
- Actual query had millions of rows running through a nested loop operation

Statistics syntax

```
CREATE STATISTICS statistics_name
    ON [ database_name.[schema_name]. | schema_name. ]table_name
    ( column_name  [ ,...n ] )
    [ WHERE <filter_predicate> ]
    [ WITH {
           FULLSCAN
           | SAMPLE number PERCENT }
    ][;]
```

```
<filter_predicate> ::=
    <conjunct> [AND <conjunct>]

<conjunct> ::=
    <disjunct> | <comparison>

<disjunct> ::=
        column_name IN (constant ,…)
```

```
<comparison> ::=
        column_name <comparison_op> constant

<comparison_op> ::=
    IS | IS NOT | = | <> | != | > | >= | !>
               | < | <= | !<
;
```

Best Practices : Statistics

Background

- Two levels of statistics exists in the MPP DWH Engine
  - SQL level – on Compute Nodes – maintained automatically
  - MPP level – cumulative statistics on Control Node's Shell DB for cost based optimization
- MPP level statistics is not maintained/created automatically

Goal

- Updated statistics for best cost based optimization

Best Practices : Statistics

Solution - Two options

1. Create statistics for all columns used in JOINs, GROUP BY, WHERE
   - Smart approach, but may not be feasible in all scenarios

2. Create statistics for all columns and all tables
   - Easy, but implies overhead

Partitioning

Partitioning Distributed Tables

- Distributed tables are already segmented by distributions
- Will further partition rows within a distribution, based on a partition function
- Enables for operations efficiency when adding, loading, dropping, and switching partitions
- Good for fast loading of an unused partition and then switching it in after loading

Partitioned Table Usage

- Same basic rules and guidelines as SQL Server SMP
- Use partition switch for large inserts, updates, and deletes
- Use metadata queries to explore partitions
  - Number of partitions and boundary values
  - Partitioning column name

Partitioned Table Management

- Provides the ability to create, merge, split, and switch partitions
- Allows DBAs to administer large tables in smaller chunks for loading, archiving, sliding windows, and more
- Optimizes the loading of large tables
- Enables re-indexing of smaller partitions
- Enables piecemeal data reorganization

Create Distributed Table With Partitions

```
CREATE TABLE myTable
   (
     id integer NOT NULL,
     lastName varchar(20),
     shipdate datetime
   )
   WITH
      ( DISTRIBUTION = HASH (id),
        CLUSTERED INDEX (shipdate),
        PARTITION (shipdate  RANGE RIGHT FOR VALUES
                    ( '1992-01-01', '1992-02-01', '1992-03-01',
                      '1992-04-01', ... )
                  )
     );
```

Partitioned Table with a Clustered Index

- May improve range queries
- Can hinder data load times more significantly on an MPP system than a SQL Server SMP system
- Page fragmentation can hinder query performance

Partitioned Table with a Clustered Columnstore Index

- May improve range queries
- Column oriented Store (not row oriented)
  - ➔ Lowest memory footprint if only a few columns of a table are used
  - ➔ Best compression Ratio

Partitioned Table with a Non-Clustered Index

- Usually not recommended
- Add only with great care and a lot of testing
- Can be potentially used for better concurrency
- Can cause lots of random I/O
  - Heap tables promote more sequential I/O, which generally works best in the MPP world as well as under concurrent workloads

Partitioning for manageability

Partition for manageability (maintenance)

- Typically on a date key (or integer surrogate)
- Typically same as clustered index key
- SWITCH partitions:
  - OUT for fast delete of history
  - IN to modify or add a specific historical slice
  - IN if data that is being loaded is used for maintaining a separate copy of load (e.g., for feeding DEV/TEST/DR – “dual loading”)

Lab:

Creating Databases and Tables

Microsoft Azure

Lab review

1. What is the purpose of schema's when used in Azure SQL Data Warehouse?
2. What is the difference between a hash and round_robin distribution?
3. Which index type should be used "judiciously"? and why?
4. Why should you updates statistics in Azure SQL Data Warehouse? And when is it best to perform the update?
5. How is partitioning different in Azure SQL Data Warehouse compared to SQL Server?

Summary

Summary

Azure SQL Data Warehouse Databases

Distributed Tables

Managing Indexes

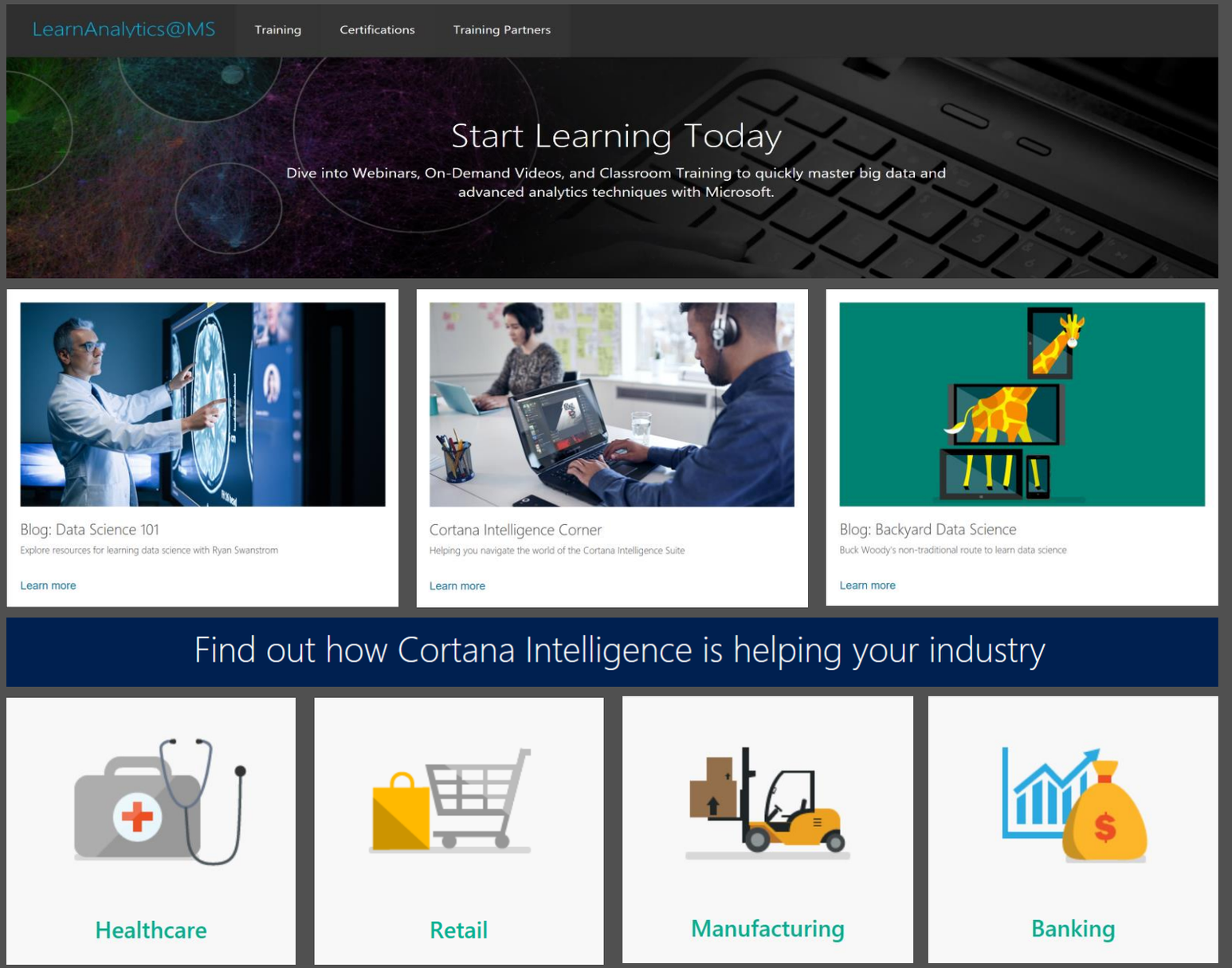
Managing Statistics

Working with Partitions

Microsoft

There are more learning options as shown in the links on the right, including:

- Online training
- Videos
- Instructor Led training
- Blogs
- Cortana Intelligence Gallery



Course Documentation

Accessing the course materials

1. Click on the picture on the left.
2. Sign in with your Live ID.
3. Look for the SQLW301 item.
4. Click on the course materials link.

Microsoft