Contents

[Introduction 1](#_Toc111555034)

[Options 2](#_Toc111555035)

[Picking the Right Engine 3](#_Toc111555036)

[Avoiding Pitfalls 4](#_Toc111555037)

[Unsupported features in Dedicated SQL Pools 6](#_Toc111555038)

[Migrating from SQL Server to Dedicated SQL Pools 7](#_Toc111555039)

## Introduction

Are you thinking about migrating to Synapse?

Are you wondering if Synapse can help alleviate some of your SQL pains?

You want to build a large database but aren’t sure of what engine to use?

If you answered Yes to any of these questions you came to the right place, but first let’s narrow the scope. In 2020 Microsoft rebranded Azure SQL Data Warehouse into Azure Synapse Analytics. This was a lot more than a name change, “Azure SQL Datawarehouse” is equivalent to what Synapse now calls “Dedicated SQL Pools”, and Synapse includes several other technologies such as Spark, Data Lake, Data Factory and Serverless SQL Pools.

In this document we will glance over some of the most popular analytic engines but will focus on Synapse Dedicated SQL Pools.

## Options

There are 4 major families of analytical database engines nowadays:

1. SMP – stands for Symmetric Parallel Processing. The main characteristic of this family is that they contain only 1 master node, which receives all read/write queries. Some offer the option to redirect selects to read-only replicas, and every query is processed entirely by a single node. Another characteristic is that each node has a copy of all the data. This family includes classic database systems such as SQL Server, Oracle, DB2, MySql and PostGress, to name a few.
2. MPP – stands for Massively Parallel Processing. The main characteristic of this family is that they contain multiple worker nodes which get parts of the data. Queries are still sent to the master node, then divided into the worker nodes for both reads and writes. This family includes Synapse Dedicated Pools, Teradata, Snowflake, IBM Netezza, GreenPlum, HP Vertica, Azure PostGress Hyperscale (citus), among others. Google BigQuery is also in this family, but it uses the serverless model, like Synapse Serverless SQL Pools.
3. HDFS – these engines rely on the hadoop distributed file system. In this architecture the data is stored in file shares and can be mounted to multiple clusters at the same time. Each cluster follows an architecture called Map-Reduce, which is like MPP. This family includes SPARK, DataBricks, HDInsight, Cloudera and HortonWorks, among others.
4. Event logs – these are variations of the Apache Lucene and Kusto indexing engines. They serve to catalog and query large volumes of telemetry or time series data. This family includes Azure Data Explorer, Elastic Search, Amason OpenSearch and InfluxDb.

## Picking the Right Engine

1. SMP are good for busy systems with high volume of small operations, such as transactional systems, but they can handle small analytical applications as well, up to a few terabytes.
2. MPP are the next natural step from SMP. They can handle up to petabytes of storage and split queries into dozens of worker nodes, while maintaining most of the relational nature of SMPs, making them easier to migrate from SQL than HDFS. These systems were designed to handle a medium amount of relatively large operations at the same time. In Synapse the maximum concurrent queries is 128. If you need to be able to process more than 128 queries at the same time look into Resultset caching, and if this feature does not resolve your problem, then Synapse is not the right choice for your workload.
3. HDFS are usually the best choice for big data workloads characterized by high velocity, volume, and variety of data. Since they can leverage the highest number of compute nodes they can handle the very large and/or fast incoming datasets. Another characteristic that allows these systems to ingest data at enormous speed is the fact that they rely on blob storage, which can not only scale massively, but also has minimal restrictions with data types, formats and constraints. Beware that there is no such thing as schemaless databases. One either enforces the schema before writes, or during reads. If no schema is enforced there is a good chance your results aren’t reliable.

These systems gained tremendous popularity in the last decade because of the huge number of open-source libraries, especially in artificial intelligence, and Databricks is arguably the most popular. They are also the go-to choose for data lake applications since they allow combining data from many different sources and bringing it to a common format.

1. Event log - Like HDFS they are well suited for high velocity, variety and volume data needs. These engines also rely on blob storage and distributed compute. If your queries require complex transformations and or combine data from multiple tables like in a data lake, you are probably better off with a HDFS engine. But if your data is more repetitive and self-describing, like application logs or IOT telemetry, you are probably better off with an event log engine, since all data is automatically indexed and partitioned based on the ingestion time, these engines provide the fastest response times.

## Avoiding Pitfalls

We typically see a lot of material on the positives sides distributed database engines, but not much on the negatives, which unfortunately leads to misuses, so we will try to describe some common challenges:

1. Limited functionality. The more an engine can scale the more challenges it faces with data consistency and programming functionality. MPPs typically implement more features frequently used in ETL compared to HDFS, so it’s generally easier to migrate from SMP to MPP than to HDFS. For instance, it’s easier to convert a stored procedure from SQL to Synapse Dedicated Pools than to a Spark notebook. Schema evolution can also be harder on the data lake, for instance delta allows adding columns, but not removing, renaming, or changing types, which are all allowed in Dedicated SQL Pools.
2. Limited security features in data lakes. It’s hard to implement complex security features such as row-level access, masking, and right to be forgotten. With Synapse dedicated pools you get these features natively like in SQL Server.
3. Low concurrency. These systems were not designed to handle a high number of concurrent operations. If you plan to have a web page or service getting invoked millions of times per day, then dedicated pools and HDFS are probably not a good fit. If you plan to use them to feed other systems, such as PowerBi or Machine Learning, a few times per day, then fine.
4. Poor consistency. There are no constraints (unique, check or referential) and no support for triggers, which means more ETL steps or the risk of poor quality in results. This is true for both dedicated SQL pools and Spark. Dedicated pools support multi table transactions, while DataBricks limits it to a single table.
5. Overuse of column stores for volatile tables.  
   Parquet files are good for storing historical data, or for ingesting append-only data, such as IOT feeds. Parquet is terrible for handling small frequent updates and deletes. To start parquet files are immutable, so to change 1 bit in 1 record you must lock and save the entire file which is often over 64mbs+. Using delta each update creates new files, and the log tracks which files are obsolete. This is what allows “time travel” but storage requirements remain high until a compaction occurs.   
   With row stores the same operation would cause 8KB write to log and to data files. While your data lake is fed once a day or every minute, that changes entirely the nature of your workload. While it is possible to implement a near real time delta lake, it is arguably wise.

Conversely the same is true about columnstore tables inside synapse dedicated sql pools. If the access pattern on a table is transactional in nature, with frequent small updates, and small selects returning many columns of a few records, avoid columnstores and use row stores instead (clustered and non-clustered indexes), like what one would do in SQL.

1. Data Movement Operations – The power or MPP and HDFS systems come from their distributed nature, conversely most of their problems also come from their distributed nature. The number one reason for poor query performance is when data needs to be copied across nodes in operations called broadcast, shuffle, partition move, etc.

The best way to prevent those is to plan carefully which tables will be replicated, and how the others will be distributed. In synapse dedicated pools you have control over which node gets which records by leveraging hash distribution, you can specify how the records will be sorted by leveraging sorted columnstores or rowstore indexes, and you can split the table in subparts using partitions.

For instance, if you join tables Order and OrderItems, and both are hash distributed on OrderNumber, the join operation would not cause a data movement because each node would have all the data for the same orders (distribution alignment). Preventing DMS is the best thing you can do to ensure faster response times. Next you have the sorting option, synapse would be able to narrow down the records faster if you apply a where/orderby on the same column used for sorting. Finally, if you partition both tables by month, and you apply filters on the partition column, then synapse would access only the needed underlying tables (partition pruning or partition elimination).

With DataLakes/Spark/Databricks one cannot specify how a table will be distributed, and there is no such thing as rowstore indexes. As consequence data movement operations are more frequent, and the optimizer relies mostly on partition pruning and zordering.

Conclusion: Classic SMP databases are still the best choice for systems with OLTP patterns such as point of sales, schools, appointments, etc. In some cases, NoSQL engines such as CosmosDb will be better, but this discussion is not in the scope of this document. What we want the reader to understand is that transitional workloads are not good candidates for Synapse Dedicated Pools or Datalake/Spark.

Synapse offers both MPP and HDFS features. MPP is covered by Dedicated SQL Pools and offers more security and performance features, as well as SQL like interface for client applications. Datalake/Spark allows processing large volumes of data faster, in greater variety, and offers an immense number of open-source libraries in several languages. If you decided to use Datalake/Spark, Synapse Serverless SQL Pools can provide a SQL like interface for the read only client applications. With Serverless SQL Pools the data still resides in the lake, and Synapse offers a table like interface using external tables and views. Finally Synapse Pipelines (aka data factory) allows writing ETL for both Datalakes and dedicated SQL pools.

For more information, please refer to the Synapse Playbook document in this repository.

## Unsupported features in Dedicated SQL Pools

* Linked servers
* Cross server queries
* Cross database queries
* Triggers
* Foreign Keys
* Sequences
* Global temporary tables
* Indexes with included columns
* Filtered indexes
* Rowstore indexes with more than 16 columns or 900 bytes
* Most query hints
* Default constraints with system functions such as getdate(), user\_name()
* Unsupported data types: 'geography', 'geometry', 'hierarchyid', 'image', 'text', 'ntext', 'sql\_variant', 'xml'.
* Primary Keys and Unique Constraints are not enforced, so duplicates can occur.  
  <https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-table-constraints>
* Identity Columns do not guarantee the order in which values are allocated.  
  <https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-tables-identity>
* Blob columns (anything(max)) are not supported in column stores, so the tables must be defined as row store. The consequences are:
  + Compression will be poor (limited to PAGE).
  + Max size of 60 TB per table and 240 TB per dedicated pool.
  + These structures do not take advantage of local cache in the compute node.

Warning: Queries over large tables containing BLOB columns (anything(max)) can severely affect overall performance of dedicated pools. Consider architecting your application in a way that stores blobs externally.

More details:  
[T-SQL feature in Synapse SQL pool - Azure Synapse Analytics | Microsoft Docs](https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/overview-features)  
<https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-tables-data-types>  
<https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-service-capacity-limits>

## Migrating from SQL Server to Dedicated SQL Pools

You read the above and think your workload can do well in a Synapse Dedicated SQL Pools. You read it supports many of the SQL features, and are curious to know which unsupported features your database uses, now do this:

1. Execute the script [Synapse Compatibility 1 Create TempDb Objects.sql] on your SQL Server Instance. You will need Sysadmin rights for this, so have a DBA do it. This script will create dozens of objects in tempdb, which will later be removed with Script 3 in this package. This script uses many systems statistics collected since the last time the server started, so for accurate results run it in production, and after the server has been running long enough to capture a good representation of your workload.
2. Execute the script [Synapse Compatibility 2 Load and Report.sql] in SQL Management Studio using rid results mode (CTRL+D before executing).   
   To prevent data truncation recent versions of SSMS 18 and later are recommended, but not required. Ensure that results in grid mode retain CR/LF and use maximum characters as high as possible. Example:

Graphical user interface, text, application

Description automatically generated

This script may take several minutes to run on large environments; however, it is not disruptive. It will load several tables in tempdb, then return many result sets. To facilitate review and sharing we recommend that you copy these results and paste them into the enclosed spreadsheet [Synapse Compatibility Checks.xlsx], tab by tab, in the same order. Make sure to select cell A2 before pasting.

Sample results:  
Graphical user interface, application

Description automatically generated

1. Execute script [Synapse Compatibility 3 Cleanup TempDb Objects.sql] to remove all objects created by 1.

Understanding Results from excel tabs:

* Instance – this tab contains high level information about the server, such as number of cores, how much memory, databases, objects, etc.
* JobSteps – this tab contains the SQL Agent Jobs, frequency, and steps. All these need to be migrated if you move to Synapse.
* Waits - this tab contains the wait statistics since last start, it serves to understand if your workload is logically or physically bound. If physically we can determine if its CPU, IO or Network.
* TopSQL – this tab contains the top stored procedures sorted by total worker time descending. It gives us an idea of what is keeping the server busy.
* Databases – this tab lists all user databases in the instance with size, object stats and a total number of incompatibilities by type. Columns in Yellow represent less concerning issues. Columns in Orange represent issues which require more effort. Most columns are self-explanatory, but for a better understanding use this reference:

|  |  |  |
| --- | --- | --- |
| Column | Comments | Details |
| Empty | Tables with no records. | Go to the Tables tab and filter on RowCount. |
| Small | Tables with less than 6 million rows, which are probably better off as clustered indexes and possibly replicated as long as they are not updated frequently. Tables larger than this are better off as distributed. | Go to the Tables tab and filter on RowCount. |
| Medium | Tables with between 6 and 60 million rows, they could be represented as row or column stores, depending on the access pattern. | Go to the Tables tab and filter on RowCount. |
| Large | tables with 60 million to 1 billion rows, and are more likely suited to hash distribution and column stores, unless they access pattern has frequent updates and small range selects. | Go to the Tables tab and filter on RowCount. |
| Very large | tables with over 1 billion rows, candidates for hash distribution as well as partitioning. | Go to the Tables tab and filter on RowCount. |
| UniqueIndexes | Unique indexes are supported but not enforced in Synapse.  So additional ETL/maintenance is needed to ensure data quality. | Go to the Indexes tab and filter on Is\_Unique=1. You may also want to filter on allocation\_desc = IN\_ROW\_DATA. |
| ColumStores | Tables that already use column compression. | Go to the Indexes tab and filter on Data\_Compression= ColumnStore. You may also want to filter on allocation\_desc = IN\_ROW\_DATA. |
| IncludedColumnIndexes | Synapse does not support indexes with included columns, so the included columns need to be removed or moved to the index columns. | Go to the Indexes tab and filter on Included not Blank. You may also want to filter on allocation\_desc = IN\_ROW\_DATA. |
| LargeTextColumns | These tables contain varchar/varbinary (max) fields, so the table cannot be represented with column store. These tables will not compress well and cannot leverage the adaptive cache. | Go to Columns tab and filter on Comment = “Can’t be Column Compressed”. |
| IncompatibleDefaults | Synapse does not support default constraints with system functions. These columns need to be updated during or post ETL. | Go to Columns tab and filter on Comment = “Incompatible Default”. |
| FilteredIndexes | Synapse does not support indexes with where conditions. The filters or indexes must be removed. | Go to the Indexes tab and apply Filter not Blank. You may also want to filter on allocation\_desc = IN\_ROW\_DATA. |
| ComputedColumns | Synapse does not support computed columns. These columns need to be updated during or post ETL. | Go to the Columns tab and filter on Comment = “Is Computed”. |
| XServerRoutines | Routines which leverage linked servers. Synapse does not support linked servers, so these routines need to be deleted, or the remote tables need to be replicated to synapse. | Go to the Routines tab and filter LinkedServers not blank. |
| XOutRoutines | Routines which leverage cross database calls, which is not supported by synapse. These reference tables need to be replicated to synapse, perhaps to different schemas. | Go to the Routines tab and filter on XOutDbs not blank. |
| XOutDbs | the databases which are referenced by routines in this database. |  |
| XInRoutines | Routines from other databases which reference this db. These tables need to be replicate from Synapse back to the source server, or all databases must be migrated in a bundle. | Go to the Routines tab and review the routines of the reference databases. |
| XInDbs | List of other databases which cross refence this. |  |
| UsesForXML | Synapse does not support this functionality. This needs to be refactored outside of Synapse. | Go to the Routines tab and filter UsesForXML=1. |
| UsesCursor | Needs to be refactored to use loops.  [Use T-SQL loops - Azure Synapse Analytics | Microsoft Docs](https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/develop-loops) | Go to the Routines tab and filter UsesCursor=1. |
| IncompatibleColumns | These data types are not supported. The values can be converted to varchar/varbinary(max), but the functionality wont be preserved such as geography, distance, hierarchy, xml, etc. | Go to the Columns tab and filter Comment = “Incompatible Type” |
| IncompatibleRoutinesByType | These routine types are not supported in synapse and the functionality needs to move to out of the database. | Go to the Routines tab and filter on IncompatibleType = 1 |
| Sequences | Synapse does not support sequences. | Go to the Sequences tab. |

* Tables – all tables in all user databases. Columns to the right show incompatibilities.
* Columns – all columns in all tables. Column Comment to the right shows incompatibilities.
* Indexes – All index structures from all tables. Some indexes will have Row\_Overflow and Blob allocations. For practical understanding these are duplicate, so you may filter all rows with Allocation Desc = In\_Row\_Data.
* Routines – all routines including definition. Columns in Orange highlight incompatibilities. Routine code may be cut down to 4k bytes depending on SSMS client settings.
* Sequences – all sequences.