Memory - In Memory

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:27 AM PST

Contents

Issue

In-memory OLTP is an option to store persistant user data in memory for improved performance. Premium and Business Critical databases come with a certain amount of memory reserved for in-memory objects, usually for tables and natively-compiled SQL code. Though in-memory OLTP offers faster performance, its size is limited; this may cause runtime errors of queries, either if Insert and Update statements are exceeding the quota, or if the SQL-internal objects related to these queries run out of memory.

The most common error scenarios are that either the customer cannot enable the feature, or that a live database runs out of memory. Typical error messages for either scenarios are:

> Msg 40536, Level 16, State 2, Line 1
> 'MEMORY_OPTIMIZED tables' is not supported in this service tier of the database. See Books Online for more details on feature support in different service tiers of Windows Azure SQL Database.

> Msg 41823, Level 16, State 109, Line 15
> Could not perform the operation because the database has reached its quota for in-memory tables. This error may be transient. Please retry the operation. See 'http://go.microsoft.com/fwlink/?LinkID=623028' for more information.

Investigation / Analysis

Troubleshooting in-memory-related issues is a four-step process:

- Identify how much in-memory space is available - this depends on the chosen service tier of the database.

- Identify how much memory is being consumed by the objects in your database or instance. See the following paragraphs in this section for monitoring options.
- Determine how memory consumption is growing and how much head room you have left. By monitoring the memory consumption periodically, you can know how the memory use is growing.
- Take action to mitigate the potential memory issues. See the "Mitigation" section further below for details.

Identify the service tier (SLO) of the database

Get this information from ASC or the `MonDmDbResourceGovernance` Kusto table (see Kusto paragraph further below).

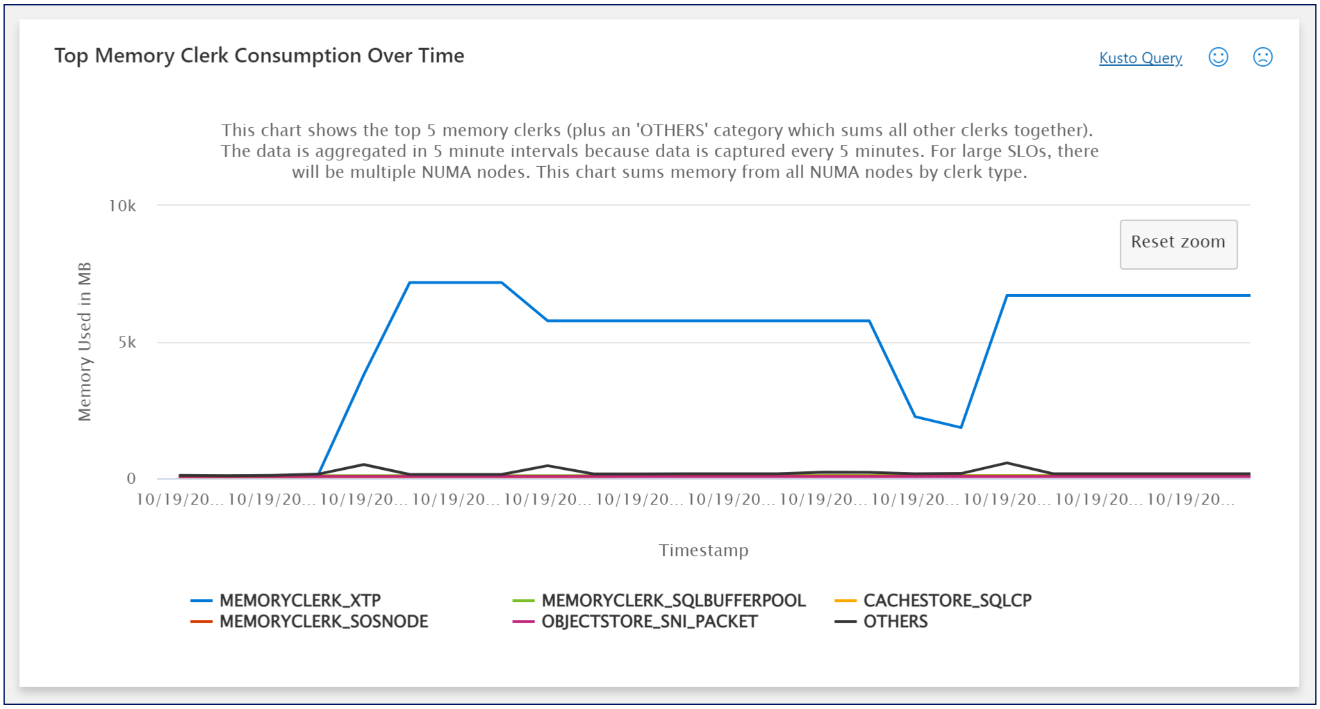
In-Memory objects are only supported in Premium (DTU model) and Business Critical (vCore model) databases. Basic, Standard, General Purpose, and Hyperscale are not supported. The available in-memory size depends on the chosen service tier.

The exact limits for each service tier are documented in the following set of articles:

- DTU model: Single databases and Elastic pools
- vCore model: Single databases and Elastic pools

ASC

ASC's "Performance - Memory" tab can show you an overview of the XTP in-memory consuption. This is the same information as in the `sys.dm_os_memory_clerks` DMV and the `MonSqlMemoryClerkStats` Kusto query - see further below.

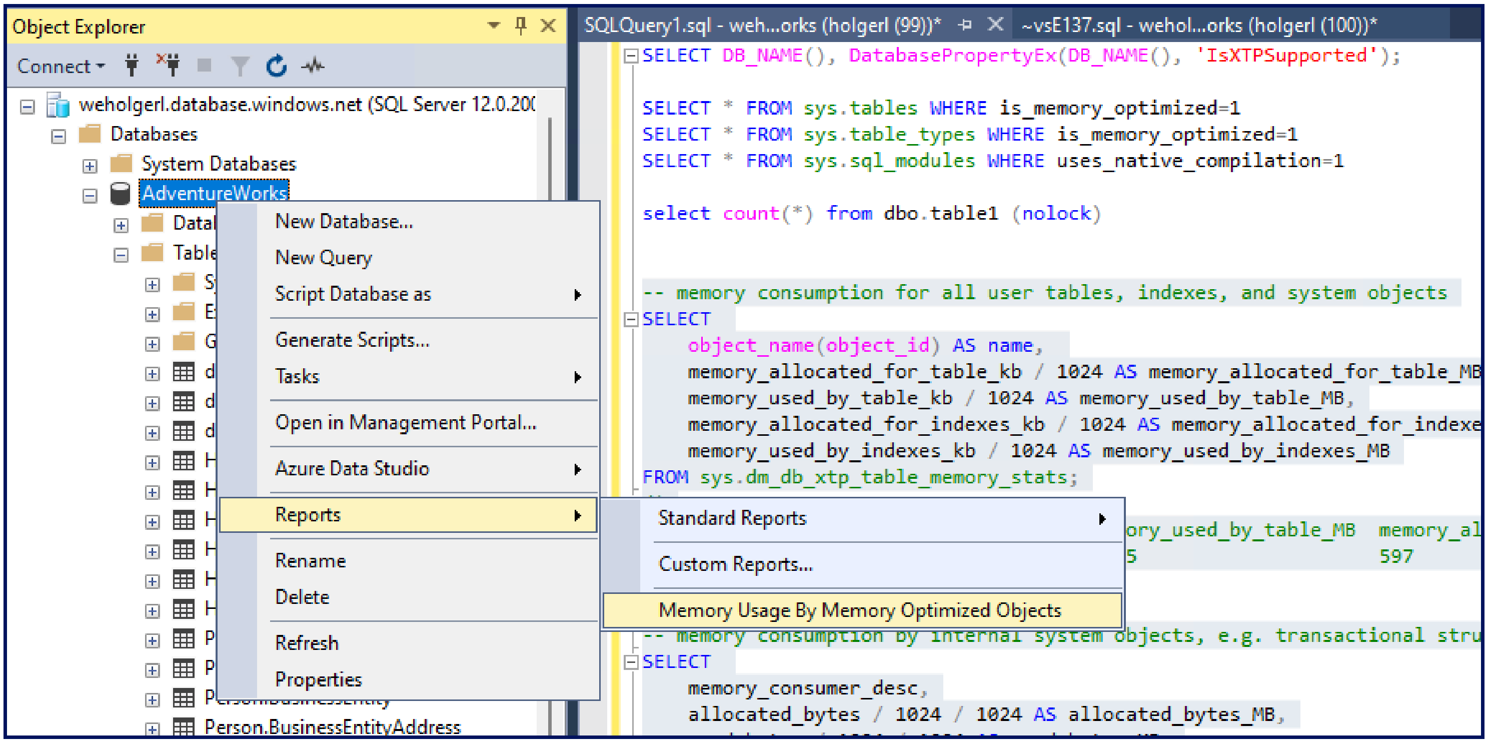


Top Memory Clerk Consumption Over Time

This chart shows the top 5 memory clerks (plus an 'OTHERS' category which sums all other clerks together). The data is aggregated in 5 minute intervals because data is captured every 5 minutes. For large SLOs, there will be multiple NUMA nodes. This chart sums memory from all NUMA nodes by clerk type.

SSMS

View the "Memory Usage By Memory Optimized Objects" standard report to get an overview about the main in-memory consumers:


Object Explorer
Connect
weholgerl.database.windows.net (SQL Server 12.0.20
Databases
System Databases
AdventureWorks
Datal
Table
Person.BusinessEntityAddress
New Database...
New Query
Script Database as
Generate Scripts...
Tasks
Open in Management Portal...
Azure Data Studio
Reports
Rename
Delete
Refresh
Properties
Standard Reports
Custom Reports...
Memory Usage By Memory Optimized Objects
SQLQuery1.sql - weh...orks (holgerl (99))*
~vsE137.sql - wehol...orks (holgerl (100))*

```
SELECT DB_NAME(), DatabasePropertyEx(DB_NAME(), 'IsXTPSupported');

SELECT * FROM sys.tables WHERE is_memory_optimized=1
SELECT * FROM sys.table_types WHERE is_memory_optimized=1
SELECT * FROM sys.sql_modules WHERE uses_native_compilation=1

select count(*) from dbo.table1 (nolock)

-- memory consumption for all user tables, indexes, and system objects
SELECT
    object_name(object_id) AS name,
    memory_allocated_for_table_kb / 1024 AS memory_allocated_for_table_MB
    memory_used_by_table_kb / 1024 AS memory_used_by_table_MB,
    memory_allocated_for_indexes_kb / 1024 AS memory_allocated_for_indexe
    memory_used_by_indexes_kb / 1024 AS memory_used_by_indexes_MB
FROM sys.dm_db_xtp_table_memory_stats;

                                   ory_used_by_table_MB  memory_al
                                   5                     597

-- memory consumption by internal system objects, e.g. transactional stru
SELECT
    memory_consumer_desc,
    allocated_bytes / 1024 / 1024 AS allocated_bytes_MB,
```

DMVs

Use the following DMVs to monitor the available memory and to get an overview which objects are using it..

sys.dm_db_xtp_table_memory_stats - user tables, indexes, and system objects

```
-- memory consumption for all user tables, indexes, and system objects
SELECT
    object_name(object_id) AS name,
    memory_allocated_for_table_kb / 1024 AS memory_allocated_for_table_MB,
    memory_used_by_table_kb / 1024 AS memory_used_by_table_MB,
    memory_allocated_for_indexes_kb / 1024 AS memory_allocated_for_indexes_MB,
    memory_used_by_indexes_kb / 1024 AS memory_used_by_indexes_MB
FROM sys.dm_db_xtp_table_memory_stats;

/*
name    memory_allocated_for_table_MB  memory_used_by_table_MB  memory_allocated_for_indexes_MB  memory_used_b
------- ------------------------------ ------------------------ -------------------------------- -------------
table1  3231                           3215                     597                              166
*/
```

sys.dm_db_resource_stats - recent resource usage

```
SELECT
    end_time,
    xtp_storage_percent,
    avg_memory_usage_percent, avg_cpu_percent, avg_data_io_percent, avg_log_write_percent
FROM sys.dm_db_resource_stats

/*
end_time                 xtp_storage_percent  avg_memory_usage_percent  avg_cpu_percent  avg_data_io_percent
------------------------ -------------------- ------------------------- ---------------- --------------------
2022-10-19 15:02:19.860  61.69                31.40                     0.00             0.00
2022-10-19 15:02:04.817  61.69                31.40                     0.00             0.00
2022-10-19 15:01:49.770  61.69                31.40                     0.00             0.00
2022-10-19 15:01:34.720  61.69                31.40                     0.00             0.00
2022-10-19 15:01:19.673  61.69                31.40                     0.00             0.00
*/
```

sys.dm_xtp_system_memory_consumers - internal system objects

```
-- memory consumption by internal system objects, e.g. transactional structures, buffers for data and delta fi
SELECT
    memory_consumer_desc,
    allocated_bytes / 1024 / 1024 AS allocated_bytes_MB,
    used_bytes / 1024 / 1024 AS used_bytes_MB,
    allocation_count
FROM sys.dm_xtp_system_memory_consumers
ORDER BY allocated_bytes DESC;

/*
memory_consumer_desc        allocated_bytes_MB  used_bytes_MB  allocation_count
--------------------------- ------------------- -------------- ----------------
Lookaside heap              703                 80             155813
System heap                 66                  0              2567
Transaction constraint set  55                  0              0
Transaction write set       27                  0              0
Transaction save-point set  4                   3              155813
Log IO proxy                2                   0              0
Transaction                 0                   0              0
(...)
*/
```

sys.dm_os_memory_objects - memory consumption at run-time

```
-- memory consumption at run-time when accessing memory-optimized tables, e.g. by the procedure cache; all run
SELECT
    type,
    memory_object_address,
    pages_in_bytes / 1024 / 1024 AS pages_in_bytes_MB,
    max_pages_in_bytes / 1024 / 1024 AS max_pages_in_bytes_MB,
    bytes_used,
    exclusive_access_count
FROM sys.dm_os_memory_objects
WHERE type LIKE '%xtp%';

/*
type                           memory_object_address  pages_in_bytes_MB  max_pages_in_bytes_MB  bytes_used  ex
------------------------------ ---------------------- ------------------ ---------------------- ----------- --
MEMOBJ_XTPDB                   0xB4FC979F35969BBF     9                  9                      NULL        53
MEMOBJ_XTPDB                   0x53E5DF654A835D76     0                  0                      NULL        13
MEMOBJ_XTPPROCCACHE            0x7334E7266F4F5250     0                  0                      NULL        0
MEMOBJ_XTPPROCPARTITIONEDHEAP  0x9632EBE34C5409CD     0                  0                      NULL        NU
MEMOBJ_XTPBLOCKALLOC           0x31284044180D338B     9                  9                      NULL        5
MEMOBJ_XTPBLOCKALLOC           0xB296D20223BF6806     3                  133                    NULL        36
MEMOBJ_XTPTHREADPOOL           0xB6198DC7C2A237C7     0                  0                      NULL        20
*/
```

sys.dm_os_memory_clerks - all memory used by the inmemory engine

According to sys.dm_os_memory_clerks (Transact-SQL), the memory clerk `MEMORYCLERK_XTP` is used for In-Memory OLTP memory allocations.

This DMV returns the same values as the "MonSqlMemoryClerkStats" Kusto table - see the Kusto section below.

```
-- memory consumed by In-Memory OLTP engine across the instance
-- Memory allocated to the In-Memory OLTP engine and the memory-optimized objects is managed the same way as a
-- The clerks of type MEMORYCLERK_XTP accounts for all the memory allocated to In-Memory OLTP engine.
-- this DMV accounts for all memory used by the inmemory engine
SELECT type,
    name,
    memory_node_id,
    pages_kb / 1024 AS pages_MB,
    virtual_memory_committed_kb / 1024 AS virtual_memory_committed_MB
FROM sys.dm_os_memory_clerks
WHERE type LIKE '%xtp%';

/*
type             name             memory_node_id  pages_MB  virtual_memory_committed_MB
---------------- ---------------- --------------- --------- ---------------------------
MEMORYCLERK_XTP  Client-Default   0               877       0
MEMORYCLERK_XTP  DB_ID_9          0               5829      0
MEMORYCLERK_XTP  Client-Default   64              0         0
*/
```

Kusto

MonDmDbHadrReplicaStates

This will show you the AppName and the NodeName of the database within the specified period, taking into account any SLO changes and failovers. The values can be used to filter some of the other Kusto tables that do not contain the database or server name.

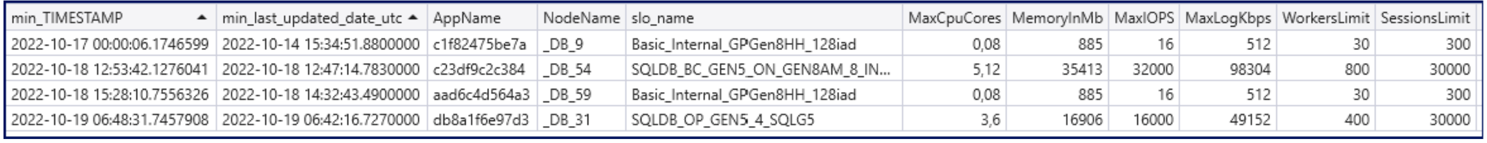
```
// Get the AppName, NodeName values
let srv = "servername";
let db = "databasename";
let startTime = datetime(2022-10-07 12:00:00Z);
let endTime = datetime(2022-10-18 23:00:00Z);
let timeRange = ago(7d);
MonDmDbHadrReplicaStates
| where  TIMESTAMP >= startTime
| where  TIMESTAMP <= endTime
// | where  TIMESTAMP >= timeRange
| where LogicalServerName =~ srv
| where logical_database_name  =~ db
| where is_primary_replica == 1
| where AppName notcontains "b-"
| summarize min(TIMESTAMP) by AppName, NodeName
| order by min_TIMESTAMP asc nulls first
```

MonDmDbResourceGovernance, MonDmDbHadrReplicaStates

This query will show you details about the scaling history, the SLO and the instance limits at a given time.

```
// Details about scaling history, SLO, resource governance, instance limits
let srv = "servername";
let db = "databasename";
let startTime = datetime(2022-10-07 12:00:00Z);
let endTime = datetime(2022-10-18 23:00:00Z);
let timeRange = ago(7d);
MonDmDbHadrReplicaStates
| where  TIMESTAMP >= startTime
| where  TIMESTAMP <= endTime
// | where  TIMESTAMP >= timeRange
| where LogicalServerName =~ srv
| where logical_database_name  =~ db
| where is_primary_replica == 1
| where AppName notcontains "b-"
| summarize min(TIMESTAMP) by AppName, NodeName
| join kind=inner
(
    MonDmDbResourceGovernance
    | where  TIMESTAMP >= startTime
    | where  TIMESTAMP <= endTime
    // | where  TIMESTAMP >= timeRange
    | where server_name =~ srv
    | where database_name =~ db
    | project TIMESTAMP, AppName, NodeName, slo_name, primary_group_max_cpu, min_cores, max_db_memory, primary
    | extend MaxCpuCores = (primary_group_max_cpu*min_cores)/100
    | extend MemoryInMb = (max_db_memory)/1024
    | extend MaxIOPS = primary_group_max_io
    | extend MaxLogKbps = primary_max_log_rate/1024.0
    | extend WorkersLimit = primary_group_max_workers
    | extend SessionsLimit = max_sessions
    | project last_updated_date_utc, AppName, NodeName, slo_name, MaxCpuCores, MemoryInMb, MaxIOPS, MaxLogKbps
    | summarize min(last_updated_date_utc) by AppName , NodeName, slo_name, MaxCpuCores, MemoryInMb, MaxIOPS,
) on AppName, NodeName
| project min_TIMESTAMP, min_last_updated_date_utc, AppName, NodeName, slo_name, MaxCpuCores, MemoryInMb, MaxI
```

Sample output:



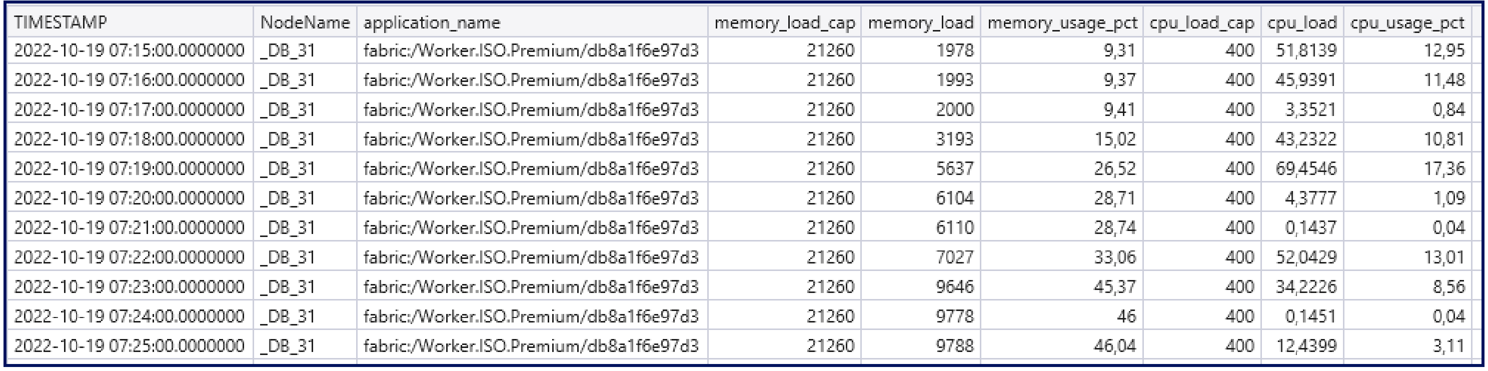
min_TIMESTAMP ▲	min_last_updated_date_utc ▲	AppName	NodeName	slo_name	MaxCpuCores	MemoryInMb	MaxIOPS	MaxLogKbps	WorkersLimit	SessionsLimit
2022-10-17 00:00:06.1746599	2022-10-14 15:34:51.8800000	c1f82475be7a	_DB_9	Basic_Internal_GPGen8HH_128iad	0,08	885	16	512	30	300
2022-10-18 12:53:42.1276041	2022-10-18 12:47:14.7830000	c23df9c2c384	_DB_54	SQLDB_BC_GEN5_ON_GEN8AM_8_IN...	5,12	35413	32000	98304	800	30000
2022-10-18 15:28:10.7556326	2022-10-18 14:32:43.4900000	aad6c4d564a3	_DB_59	Basic_Internal_GPGen8HH_128iad	0,08	885	16	512	30	300
2022-10-19 06:48:31.7457908	2022-10-19 06:42:16.7270000	db8a1f6e97d3	_DB_31	SQLDB_OP_GEN5_4_SQLG5	3,6	16906	16000	49152	400	30000

MonRgLoad

This query shows you the memory usage and its cap (maximum available memory) over time.

```
// MonRgLoad shows memory usage and memory cap (max available memory)
let srv = "servername";
let db = "databasename";
let startTime = datetime(2022-10-19 07:00:00Z);
let endTime = datetime(2022-10-19 23:00:00Z);
let timeRange = ago(7d);
let AppNames = MonAnalyticsDBSnapshot
    | where  TIMESTAMP >= startTime
    | where  TIMESTAMP <= endTime
    // | where  TIMESTAMP >= timeRange
    | where logical_server_name =~ srv
    | where logical_database_name =~ db
    | extend AppName = sql_instance_name
    | distinct AppName;
MonRgLoad
| where  TIMESTAMP >= startTime
| where  TIMESTAMP <= endTime
// | where  TIMESTAMP >= timeRange
| where event == "instance_load"
| where application_name has_any (AppNames)
| where NodeName in ("_DB_54", "_DB_9", "_DB_31", "_DB_59")
| where code_package_name == "Code"
| extend memory_usage_pct = round((memory_load/memory_load_cap)*100, 2)
| extend cpu_usage_pct = round((cpu_load/cpu_load_cap)*100, 2)
| project TIMESTAMP, NodeName, application_name, memory_load_cap, toint(memory_load) , memory_usage_pct, cpu_l
| order by TIMESTAMP asc
//| render timechart
```

Sample output (workload created by the sample script in More Information below):



TIMESTAMP	NodeName	application_name	memory_load_cap	memory_load	memory_usage_pct	cpu_load_cap	cpu_load	cpu_usage_pct
2022-10-19 07:15:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	1978	9,31	400	51,8139	12,95
2022-10-19 07:16:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	1993	9,37	400	45,9391	11,48
2022-10-19 07:17:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	2000	9,41	400	3,3521	0,84
2022-10-19 07:18:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	3193	15,02	400	43,2322	10,81
2022-10-19 07:19:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	5637	26,52	400	69,4546	17,36
2022-10-19 07:20:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	6104	28,71	400	4,3777	1,09
2022-10-19 07:21:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	6110	28,74	400	0,1437	0,04
2022-10-19 07:22:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	7027	33,06	400	52,0429	13,01
2022-10-19 07:23:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	9646	45,37	400	34,2226	8,56
2022-10-19 07:24:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	9778	46	400	0,1451	0,04
2022-10-19 07:25:00.0000000	_DB_31	fabric:/Worker.ISO.Premium/db8a1f6e97d3	21260	9788	46,04	400	12,4399	3,11

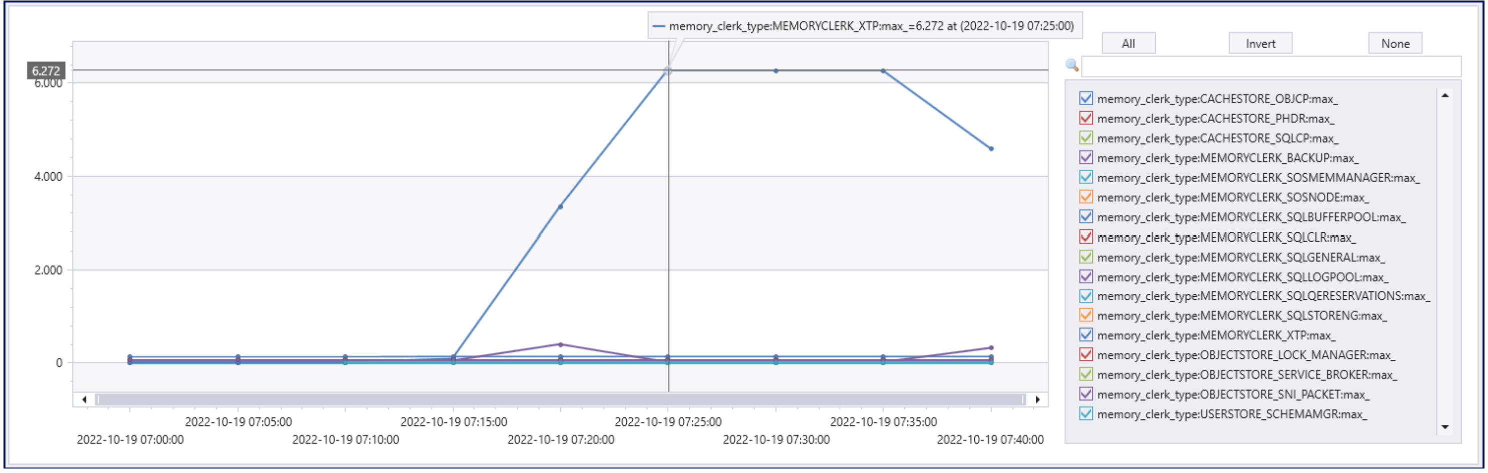
MonSqlMemoryClerkStats

According to sys.dm_os_memory_clerks (Transact-SQL) ↗, the memory clerk `MEMORYCLERK_XTP` is used for In-Memory OLTP ↗ memory allocations. See sys.dm_os_memory_clerks (Transact-SQL) ↗ for a list of memory clerks and their meaning.

This Kusto query returns the same values as the "sys.dm_os_memory_clerks" DMV - see the DMV section above.

```
let srv = "servername";
let db = "databasename";
let startTime = datetime(2022-10-19 07:00:00Z);
let endTime = datetime(2022-10-19 23:00:00Z);
let timeRange = ago(7d);
MonDmDbHadrReplicaStates
| where  TIMESTAMP >= startTime
| where  TIMESTAMP <= endTime
// | where  TIMESTAMP >= timeRange
| where LogicalServerName =~ srv
| where logical_database_name  =~ db
| where is_primary_replica == 1
| where AppName notcontains "b-"
| summarize min(TIMESTAMP) by AppName, NodeName
| join kind=inner
(
    MonSqlMemoryClerkStats
    | where  TIMESTAMP >= startTime
    | where  TIMESTAMP <= endTime
    // | where  TIMESTAMP >= timeRange
    | where LogicalServerName =~ srv
    | extend memory_MB = round(pages_kb/1024.0)
    | extend vm_committed_MB = round(vm_committed_kb/1024.0)
    | where ['memory_clerk_type'] in ("MEMORYCLERK_XTP")
    | where memory_clerk_type !in~ ("MEMORYCLERK_XE_BUFFER")
    //| where ['memory_clerk_type'] in ("MEMORYCLERK_SQLCLR", "MEMORYCLERK_SQLBUFFERPOOL", "CACHESTORE_SQLCP",
    | project TIMESTAMP, NodeName, LogicalServerName, AppName, memory_clerk_type, memory_clerk_name, memory_MB
) on AppName, NodeName
| summarize memory_MB = sum(memory_MB) + sum(vm_committed_MB) by ['memory_clerk_type'], bin(TIMESTAMP, 5min)
| render timechart
```

Sample output (workload created by the sample script in [More Information](#) below):



```
AppName       NodeName  min_TIMESTAMP               TIMESTAMP                   memory_clerk_type  memory_cle
------------- --------- --------------------------- --------------------------- ------------------ ----------
db8a1f6e97d3  _DB_31    2022-10-19 07:00:31.4532951 2022-10-19 08:41:17.0215645  MEMORYCLERK_XTP    Client-Def
db8a1f6e97d3  _DB_31    2022-10-19 07:00:31.4532951 2022-10-19 08:41:17.0215645  MEMORYCLERK_XTP    DB_ID_9
db8a1f6e97d3  _DB_31    2022-10-19 07:00:31.4532951 2022-10-19 08:46:18.0702274  MEMORYCLERK_XTP    Client-Def
db8a1f6e97d3  _DB_31    2022-10-19 07:00:31.4532951 2022-10-19 08:46:18.0702274  MEMORYCLERK_XTP    DB_ID_9
db8a1f6e97d3  _DB_31    2022-10-19 07:00:31.4532951 2022-10-19 08:51:19.1189697  MEMORYCLERK_XTP    Client-Def
db8a1f6e97d3  _DB_31    2022-10-19 07:00:31.4532951 2022-10-19 08:51:19.1189697  MEMORYCLERK_XTP    DB_ID_9
```

Mitigation

Error 40536 'MEMORY_OPTIMIZED tables' is not supported in this service tier

- Scale the database to Premium or Business Critical if the customer is on Basic, Standard, or General Purpose.
- If the current SLO is Hyperscale: Consider staying with Hyperscale, as there are other factors that may prohibit from moving to Premium or Business Critical (e.g. storage size, redundant compute nodes).

Error 41823 Database has reached its quota for in-memory tables

- Consider moving one or more in-memory objects to a storage-bound object to free some of the in-memory space. This requires to recreate the object without the `MEMORY_OPTIMIZED = ON` option and copying over the data from the in-memory object.
- Consider deleting some data from in-memory objects, if applicable from an application perspective.
- If a columnstore index is involved, also see Unable to release memory used for inmemory table.
- Scale to a higher performance level of Premium or Business Critical. Check the articles below for the next-higher in-memory size.

The exact resource limits for each service tier are documented in the following set of articles:

- DTU model: Single databases and Elastic pools
- vCore model: Single databases and Elastic pools

More Information

There are no specific prerequisites for using in-memory OLTP objects in Azure SQL Database, except for being supported only in Premium and Business Critical service tiers. There is no need to create separate file groups or anything storage-related; you just create the objects and go for it.

Here is a very simple SQL script to create and fill an in-memory table up to the SLO limit:

```
-- create memory-optimized table
CREATE TABLE dbo.table1
(
    c1 INT IDENTITY PRIMARY KEY NONCLUSTERED,
    c2 NVARCHAR(100)
) WITH (MEMORY_OPTIMIZED = ON)
GO

set nocount on;

INSERT INTO dbo.table1 (c2) VALUES ('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

DECLARE @i INT = 1
WHILE @i < 100
BEGIN
    INSERT INTO dbo.table1 (c2) SELECT c2 from dbo.table1;
    SET @i += 1
    waitfor delay '00:00:10'
END;

DELETE dbo.table1;

DROP TABLE db. table1;
```

Public Doc Reference

- Optimize performance by using in-memory technologies in Azure SQL Database and Azure SQL Managed Instance
- Determining if a Table or Stored Procedure Should Be Ported to In-Memory OLTP
- In-Memory OLTP in Azure SQL Database
- In-Memory OLTP

How good have you found this content?

🙂 🙁