Table of Contents

System Dynamic Management Views

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Dynamic management views and functions return server state information that can be used to monitor the health of a server instance, diagnose problems, and tune performance.

> **IMPORTANT**
>
> Dynamic management views and functions return internal, implementation-specific state data. Their schemas and the data they return may change in future releases of SQL Server. Therefore, dynamic management views and functions in future releases may not be compatible with the dynamic management views and functions in this release. For example, in future releases of SQL Server, Microsoft may augment the definition of any dynamic management view by adding columns to the end of the column list. We recommend against using the syntax `SELECT * FROM dynamic_management_view_name` in production code because the number of columns returned might change and break your application.

There are two types of dynamic management views and functions:

- Server-scoped dynamic management views and functions. These require VIEW SERVER STATE permission on the server.
- Database-scoped dynamic management views and functions. These require VIEW DATABASE STATE permission on the database.

Querying Dynamic Management Views

Dynamic management views can be referenced in Transact-SQL statements by using two-part, three-part, or four-part names. Dynamic management functions on the other hand can be referenced in Transact-SQL statements by using either two-part or three-part names. Dynamic management views and functions cannot be referenced in Transact-SQL statements by using one-part names.

All dynamic management views and functions exist in the sys schema and follow this naming convention dm_*. When you use a dynamic management view or function, you must prefix the name of the view or function by using the sys schema. For example, to query the dm_os_wait_stats dynamic management view, run the following query:

```
SELECT wait_type, wait_time_ms
FROM sys.dm_os_wait_stats;
```

Required Permissions

To query a dynamic management view or function requires SELECT permission on object and VIEW SERVER STATE or VIEW DATABASE STATE permission. This lets you selectively restrict access of a user or login to dynamic management views and functions. To do this, first create the user in master and then deny the user SELECT permission on the dynamic management views or functions that you do not want them to access. After this, the user cannot select from these dynamic management views or functions, regardless of database context of the user.

NOTE

Because DENY takes precedence, if a user has been granted VIEW SERVER STATE permissions but denied VIEW DATABASE STATE permission, the user can see server-level information, but not database-level information.

In This Section

Dynamic management views and functions have been organized into the following categories.

Always On Availability Groups Dynamic Management Views and Funtions (Transact-SQL)	Memory-Optimized Table Dynamic Management Views (Transact-SQL)
Change Data Capture Related Dynamic Management Views (Transact-SQL)	Object Related Dynamic Management Views and Functions (Transact-SQL)
Change Tracking Related Dynamic Management Views	Query Notifications Related Dynamic Management Views (Transact-SQL)
Common Language Runtime Related Dynamic Management Views (Transact-SQL)	Replication Related Dynamic Management Views (Transact-SQL)
Database Mirroring Related Dynamic Management Views (Transact-SQL)	Resource Governor Related Dynamic Management Views (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)	Security-Related Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)	Server-Related Dynamic Management Views and Functions (Transact-SQL)
Extended Events Dynamic Management Views	Service Broker Related Dynamic Management Views (Transact-SQL)
Filestream and FileTable Dynamic Management Views (Transact-SQL)	Spatial Data Related Dynamic Management Views and Functions (Transact-SQL)
Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)	SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)
Geo-Replication Dynamic Management Views and Functions (Azure SQL Database)	SQL Server Operating System Related Dynamic Management Views (Transact-SQL)
Index Related Dynamic Management Views and Functions (Transact-SQL)	Stretch Database Dynamic Management Views (Transact-SQL)
I O Related Dynamic Management Views and Functions (Transact-SQL)	Transaction Related Dynamic Management Views and Functions (Transact-SQL)

See Also

GRANT Server Permissions (Transact-SQL)
GRANT Database Permissions (Transact-SQL)

System Views (Transact-SQL)

Always On Availability Groups Dynamic Management Views - Functions

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the dynamic management views and functions that are related to Always On availability groups.

In This Section

sys.dm_hadr_auto_page_repair	sys.dm_hadr_cluster_networks
sys.dm_hadr_availability_group_states	sys.dm_hadr_database_replica_cluster_states
sys.dm_hadr_availability_replica_cluster_nodes	sys.dm_hadr_database_replica_states
sys.dm_hadr_availability_replica_cluster_states	sys.dm_hadr_instance_node_map
sys.dm_hadr_availability_replica_states	sys.dm_hadr_name_id_map
sys.dm_hadr_cluster	sys.dm_tcp_listener_states
sys.dm_hadr_cluster_members	

See Also

AlwaysOn Availability Groups (SQL Server)
AlwaysOn Availability Groups Catalog Views (Transact-SQL)
Monitor Availability Groups (Transact-SQL)

sys.dm_hadr_auto_page_repair (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for every automatic page-repair attempt on any availability database on an availability replica that is hosted for any availability group by the server instance. This view contains rows for the latest automatic page-repair attempts on a given primary or secondary database, with a maximum of 100 rows per database. As soon as a database reaches the maximum, the row for its next automatic page-repair attempt replaces one of the existing entries.

The following table defines the meaning of the various columns:

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	ID of the database to which this row corresponds.
file_id	**int**	ID of the file in which the page is located.
page_id	**bigint**	ID of the page in the file.
error_type	**int**	Type of the error. The values can be: -1 = All hardware 823 errors 1 = 824 errors other than a bad checksum or a torn page (such as a bad page ID) 2 = Bad checksum 3 = Torn page

COLUMN NAME	DATA TYPE	DESCRIPTION
page_status	**int**	The status of the page-repair attempt: 2 = Queued for request from partner. 3 = Request sent to partner. 4 = Queued for automatic page repair (response received from partner). 5 = Automatic page repair succeeded and the page should be usable. 6 = Irreparable. This indicates that an error occurred during page-repair attempt, for example, because the page is also corrupted on the partner, the partner is disconnected, or a network problem occurred. This state is not terminal; if corruption is encountered again on the page, the page will be requested again from the partner.
modification_time	**datetime**	Time of last change to the page status.

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Automatic Page Repair (Availability Groups: Database Mirroring)
suspect_pages (Transact-SQL)
Manage the suspect_pages Table (SQL Server)

sys.dm_hadr_availability_group_states (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each Always On availability group that possesses an availability replica on the local instance of SQL Server. Each row displays the states that define the health of a given availability group.

> **NOTE**
> To obtain the complete list of, query the sys.availability_groups catalog view.

COLUMN NAME	DATA TYPE	DESCRIPTION
group_id	**uniqueidentifier**	Unique identifier of the availability group.
primary_replica	**varchar(128)**	Name of the server instance that is hosting the current primary replica. NULL = Not the primary replica or unable to communicate with the WSFC failover cluster.
primary_recovery_health	**tinyint**	Indicates the recovery health of the primary replica, one of: 0 = In progress 1 = Online NULL On secondary replicas the **primary_recovery_health** column is NULL.
primary_recovery_health_desc	**nvarchar(60)**	Description of **primary_replica_health**, one of: ONLINE_IN_PROGRESS ONLINE NULL

COLUMN NAME	DATA TYPE	DESCRIPTION
secondary_recovery_health	**tinyint**	Indicates the recovery health of a secondary replica replica,one of: 0 = In progress 1 = Online NULL On the primary replica, the **secondary_recovery_health** column is NULL.
secondary_recovery_health_desc	**nvarchar(60)**	Description of **secondary_recovery_health**, one of: ONLINE_IN_PROGRESS ONLINE NULL
synchronization_health	**tinyint**	Reflects a rollup of the **synchronization_health** of all availability replicas in the availability group. Below are the possible values and their descriptions. 0: Not healthy. None of the availability replicas have a healthy **synchronization_health** (2 = HEALTHY). 1: Partially healthy. The synchronization health of some, but not all, availability replicas is healthy. 2: Healthy. The synchronization health of every availability replica is healthy. For information about replica synchronization health, see the **synchronization_health** column in sys.dm_hadr_availability_replica_states (Transact-SQL).
synchronization_health_desc	**nvarchar(60)**	Description of **synchronization_health**, one of: NOT_HEALTHY PARTIALLY_HEALTHY HEALTHY

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Monitor Availability Groups (Transact-SQL)
Always On Availability Groups (SQL Server)
Always On Availability Groups Dynamic Management Views and Functions (Transact-SQL)

sys.dm_hadr_availability_replica_cluster_nodes (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for every availability replica (regardless of join state) of the Always On availability groups in the Windows Server Failover Clustering (WSFC) cluster.

COLUMN NAME	DATA TYPE	DESCRIPTION
group_name	**nvarchar(256)**	Name of the availability group.
replica_server_name	**nvarchar(256)**	Name of the instance of SQL Server hosting the replica.
node_name	**nvarchar(256)**	Name of the cluster node.

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Monitor Availability Groups (Transact-SQL)
Overview of Always On Availability Groups (SQL Server)

sys.dm_hadr_availability_replica_cluster_states (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each Always On availability replica (regardless of its join state) of all Always On availability groups (regardless of replica location) in the Windows Server Failover Clustering (WSFC) cluster.

COLUMN NAME	DATA TYPE	DESCRIPTION
replica_id	**uniqueidentifier**	Unique identifier of the availability replica.
replica_server_name	**nvarchar(256)**	Name of the instance of SQL Server hosting the replica.
group_id	**uniqueidentifier**	Unique identifier of the availability group.
join_state	**tinyint**	0 = Not joined 1 = Joined, standalone instance 2 = Joined, failover cluster instance
join_state_desc	**nvarchar(60)**	NOT_JOINED JOINED_STANDALONE_INSTANCE JOINED_FAILOVER_CLUSTER_INSTANCE

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Monitor Availability Groups (Transact-SQL)

sys.dm_hadr_availability_replica_states (Transact-SQL)

5/3/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each local replica and a row for each remote replica in the same Always On availability group as a local replica. Each row contains information about the state of a given replica.

> **IMPORTANT**
> To obtain information about every replica in a given availability group, query **sys.dm_hadr_availability_replica_states** on the server instance that is hosting the primary replica. When queried on a server instance that is hosting a secondary replica of an availability group, this dynamic management view returns only local information for the availability group.

COLUMN NAME	DATA TYPE	DESCRIPTION
replica_id	**uniqueidentifier**	Unique identifier of the replica.
group_id	**uniqueidentifier**	Unique identifier of the availability group.
is_local	**bit**	Whether the replica is local, one of: 0 = Indicates a remote secondary replica in an availability group whose primary replica is hosted by the local server instance. This value occurs only on the primary replica location. 1 = Indicates a local replica. On secondary replicas, this is the only available value for the availability group to which the replica belongs.
role	**tinyint**	Current Always On availability groups role of a local replica or a connected remote replica, one of: 0 = Resolving 1 = Primary 2 = Secondary For information about Always On availability groups roles, see Overview of Always On Availability Groups (SQL Server).

COLUMN NAME	DATA TYPE	DESCRIPTION
role_desc	**nvarchar(60)**	Description of **role**, one of: RESOLVING PRIMARY SECONDARY
operational_state	**tinyint**	Current operational state of the replica, one of: 0 = Pending failover 1 = Pending 2 = Online 3 = Offline 4 = Failed 5 = Failed, no quorum NULL = Replica is not local. For more information, see Roles and Operational States, later in this topic.
operational_state_desc	**nvarchar(60)**	Description of **operational_state**, one of: PENDING_FAILOVER PENDING ONLINE OFFLINE FAILED FAILED_NO_QUORUM NULL

COLUMN NAME	DATA TYPE	DESCRIPTION
recovery_health	**tinyint**	Rollup of the **database_state** column of the sys.dm_hadr_database_replica_states dynamic management view. The following are the possible values and their descriptions. 0 : In progress. At least one joined database has a database state other than ONLINE (**database_state** is not 0). 1 : Online. All the joined databases have a database state of ONLINE (**database_state** is 0). NULL : **is_local** = 0
recovery_health_desc	**nvarchar(60)**	Description of **recovery_health**, one of: ONLINE_IN_PROGRESS ONLINE NULL
synchronization_health	**tinyint**	Reflects a rollup of the database synchronization state (**synchronization_state**)of all joined availability databases (also known as *replicas*) and the availability mode of the replica (synchronous-commit or asynchronous-commit mode). The rollup will reflect the least healthy accumulated state the databases on the replica. Below are the possible values and their descriptions. 0 : Not healthy. At least one joined database is in the NOT SYNCHRONIZING state. 1 : Partially healthy. Some replicas are not in the target synchronization state: synchronous-commit replicas should be synchronized, and asynchronous-commit replicas should be synchronizing. 2 : Healthy. All replicas are in the target synchronization state: synchronous-commit replicas are synchronized, and asynchronous-commit replicas are synchronizing.

COLUMN NAME	DATA TYPE	DESCRIPTION
synchronization_health_desc	**nvarchar(60)**	Description of **synchronization_health**, one of: NOT_HEALTHY PARTIALLY_HEALTHY HEALTHY
connected_state	**tinyint**	Whether a secondary replica is currently connected to the primary replica. The possible values are shown below with their descriptions. 0 : Disconnected. The response of an availability replica to the DISCONNECTED state depends on its role: On the primary replica, if a secondary replica is disconnected, its secondary databases are marked as NOT SYNCHRONIZED on the primary replica, which waits for the secondary to reconnect; On a secondary replica, upon detecting that it is disconnected, the secondary replica attempts to reconnect to the primary replica. 1 : Connected. Each primary replica tracks the connection state for every secondary replica in the same availability group. Secondary replicas track the connection state of only the primary replica.
connected_state_desc	**nvarchar(60)**	Description of **connection_state**, one of: DISCONNECTED CONNECTED
last_connect_error_number	**int**	Number of the last connection error.
last_connect_error_description	**nvarchar(1024)**	Text of the **last_connect_error_number** message.
last_connect_error_timestamp	**datetime**	Date and time timestamp indicating when the **last_connect_error_number** error occurred.

Roles and Operational States

The role, **role**, reflects the state of a given availability replica and the operational state, **operational_state**, describes whether the replica is ready to process client requests for all the database of the availability replica. The following is a summary of the operational states that are possible for each role: RESOLVING, PRIMARY, and SECONDARY.

RESOLVING: When an availability replica is in the RESOLVING role, the possible operational states are as shown in the following table.

OPERATIONAL STATE	DESCRIPTION
PENDING_FAILOVER	A failover command is being processed for the availability group.
OFFLINE	All configuration data for the availability replica has been updated on WSFC cluster and, also, in local metadata, but the availability group currently lacks a primary replica.
FAILED	A read failure has occurred during an attempt trying to retrieve information from the WSFC cluster.
FAILED_NO_QUORUM	The local WSFC node does not have quorum. This is an inferred state.

PRIMARY: When an availability replica is performing the PRIMARY role, it is currently the primary replica. The possible operational states are as shown in the following table.

OPERATIONAL STATE	DESCRIPTION
PENDING	This is a transient state, but a primary replica can be stuck in this state if workers are not available to process requests.
ONLINE	The availability group resource is online, and all database worker threads have been picked up.
FAILED	The availability replica is unable to read to and/or write from the WSFC cluster.

SECONDARY: When an availability replica is performing the SECONDARY role, it is currently a secondary replica. The possible operational states are as shown in the table below.

OPERATIONAL STATE	DESCRIPTION
ONLINE	The local secondary replica is connected to the primary replica.
FAILED	The local secondary replica is unable to read to and/or write from the WSFC cluster.
NULL	On a primary replica, this value is returned when the row relates to a secondary replica.

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Overview of Always On Availability Groups (SQL Server)
Monitor Availability Groups (Transact-SQL)

sys.dm_hadr_cluster (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

If the Windows Server Failover Clustering (WSFC) node that hosts an instance of SQL Server that is enabled for Always On availability groups has WSFC quorum, **sys.dm_hadr_cluster** returns a row that exposes the cluster name and information about the quorum. If the WSFC node has no quorum, no row is returned.

> **TIP**
>
> Beginning in SQL Server 2014 (12.x), this dynamic management view supports Always On Failover Cluster Instances in addition to Always On Availability Groups.

COLUMN NAME	DATA TYPE	DESCRIPTION
cluster_name	**nvarchar(128)**	Name of the WSFC cluster that hosts the instances of SQL Server that are enabled for Always On availability groups.
quorum_type	**tinyint**	Type of quorum used by this WSFC cluster, one of: 0 = Node Majority. This quorum configuration can sustain failures of half the nodes (rounding up) minus one. For example, on a seven node cluster, this quorum configuration can sustain three node failures. 1 = Node and Disk Majority. If the disk witness remains on line, this quorum configuration can sustain failures of half the nodes (rounding up). For example, a six node cluster in which the disk witness is online could sustain three node failures. If the disk witness goes offline or fails, this quorum configuration can sustain failures of half the nodes (rounding up) minus one. For example, a six node cluster with a failed disk witness could sustain two (3-1=2) node failures. 2 = Node and File Share Majority. This quorum configuration works in a similar way to Node and Disk Majority, but uses a file-share witness instead of a disk witness. 3 = No Majority: Disk Only. If the quorum disk is online, this quorum configuration can sustain failures of all nodes except one.

COLUMN NAME	DATA TYPE	DESCRIPTION
quorum_type_desc	**varchar(50)**	Description of **quorum_type**, one of: NODE_MAJORITY NODE_AND_DISK_MAJORITY NODE_AND_FILE_SHARE_MAJORITY NO_MAJORITY:_DISK_ONLY
quorum_state	**tinyint**	State of the WSFC quorum, one of: 0 = Unknown quorum state 1 = Normal quorum 2 = Forced quorum
quorum_state_desc	**varchar(50)**	Description of **quorum_state**, one of: UNKNOWN_QUORUM_STATE NORMAL_QUORUM FORCED_QUORUM

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Always On Availability Groups Dynamic Management Views and Functions (Transact-SQL)
Always On Availability Groups Catalog Views (Transact-SQL)
Monitor Availability Groups (Transact-SQL)
sys.dm_hadr_cluster_members (Transact-SQL)

sys.dm_hadr_cluster_members (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

If the WSFC node that hosts a local instance of SQL Server that is enabled for Always On availability groups has WSFC quorum, returns a row for each of the members that constitute the quorum and the state of each of them. This includes of all nodes in the cluster (returned with CLUSTER_ENUM_NODE type by the **Clusterenum** function) and the disk or file-share witness, if any. The row returned for a given member contains information about the state of that member. For example, for a five node cluster with majority node quorum in which one node is down, when **sys.dm_hadr_cluster_members** is queried from a server instance that is that is enabled for Always On availability groups that resides on a node with quorum, **sys.dm_hadr_cluster_members** reflects the state of the down node as "NODE_DOWN".

If the WSFC node has no quorum, no rows are returned.

Use this dynamic management view to answer the following questions:

- What nodes are currently running on the WSFC cluster?
- How many more failures can the WSFC cluster tolerate before losing quorum in a majority-node case?

> **TIP**
>
> Beginning in SQL Server 2014 (12.x), this dynamic management view supports Always On Failover Cluster Instances in addition to Always On Availability Groups.

COLUMN NAME	DATA TYPE	DESCRIPTION
member_name	**nvarchar(128)**	Member name, which can be a computer name, a drive letter, or a file share path.
member_type	**tinyint**	The type of member, one of: 0 = WSFC node 1 = Disk witness 2 = File share witness
member_type_desc	**nvarchar(50)**	Description of **member_type**, one of: CLUSTER_NODE DISK_WITNESS FILE_SHARE_WITNESS

COLUMN NAME	DATA TYPE	DESCRIPTION
member_state	**tinyint**	The member state, one of: 0 = Offline 1 = Online
member_state_desc	**nvarchar(60)**	Description of **member_state**, one of: UP DOWN
number_of_quorum_votes	**tinyint**	Number of quorum votes possessed by this quorum member. For No Majority: Disk Only quorums, this value defaults to 0. For other quorum types, this value defaults to 1.

Permissions

Requires VIEW SERVER STATE permission on the server.

Examples

See Also

Always On Availability Groups Dynamic Management Views and Functions (Transact-SQL)
Always On Availability Groups Catalog Views (Transact-SQL)
Monitor Availability Groups (Transact-SQL)
AlwaysOn Availability Groups (SQL Server)

sys.dm_hadr_cluster_networks (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for every WSFC cluster member that is participating in an availability group's subnet configuration. You can use this dynamic management view to validate the network virtual IP that is configured for each availability replica.

Primary key: **member_name** + **network_subnet_IP** + **network_subnet_prefix_length**

> **TIP**
>
> Beginning in SQL Server 2014 (12.x), this dynamic management view supports Always On Failover Cluster Instances in addition to Always On Availability Groups.

COLUMN NAME	DATA TYPE	DESCRIPTION
member_name	**nvarchar(128)**	A computer name of a node in the WSFC cluster.
network_subnet_ip	**nvarchar(48)**	Network IP address of the subnet to which the computer belongs. This can be an IPv4 or IPv6 address.
network_subnet_ipv4_mask	**nvarchar(45)**	Network subnet mask that specifies the subnet to which the IP address belongs. **network_subnet_ipv4_mask** to specify the DHCP <network_subnet_option> options in a WITH DHCP clause of the CREATE AVAILABILITY GROUP or ALTER AVAILABILITY GROUP Transact-SQL statement. NULL = IPv6 subnet.
network_subnet_prefix_length	**int**	Network IP prefix length that specifies the subnet to which the computer belongs.
is_public	**bit**	Whether the network is private or public on the WSFC cluster, one of: 0 = Private 1 = Public

COLUMN NAME	DATA TYPE	DESCRIPTION
is_ipv4	**bit**	Type of the subnet, one of: 1 = IPv4 0 = IPv6

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Failover Clustering and Always On Availability Groups (SQL Server)
Monitor Availability Groups (Transact-SQL)
sys.dm_os_cluster_nodes (Transact-SQL)
Querying the SQL Server System Catalog FAQ
Catalog Views (Transact-SQL)

sys.dm_hadr_database_replica_cluster_states (Transact-SQL)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row containing information intended to provide you with insight into the health of the availability databases in the Always On availability groups in each Always On availability group on the Windows Server Failover Clustering (WSFC) cluster. Query **sys.dm_hadr_database_replica_states** to answer the following questions:

- Are all databases in an availability group ready for a failover?
- After a forced failover, has a secondary database suspended itself locally and acknowledged its suspended state to the new primary replica?
- If the primary replica is currently unavailable, which secondary replica would allow the minimum data loss if it becomes the primary replica?
- When the value of the sys.databases **log_reuse_wait_desc** column is "AVAILABILITY_REPLICA", which secondary replica in an availability group is holding up log truncation on a given primary database?

COLUMN NAME	DATA TYPE	DESCRIPTION
replica_id	**uniqueidentifier**	Identifier of the availability replica within the availability group.
group_database_id	**uniqueidentifier**	Identifier of the database within the availability group. This identifier is identical on every replica to which this database is joined.
database_name	**sysname**	Name of a database that belongs to the availability group.
is_failover_ready	**bit**	Indicates whether the secondary database is synchronized with the corresponding primary database. one of: 0 = The database is not marked as synchronized in the cluster. The database is not ready for a failover. 1 = The database is marked as synchronized in the cluster. The database is ready for a failover.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_pending_secondary_suspend	**bit**	Indicates whether, after a forced failover, the database is pending suspension, one of: 0 = Any states except for HADR_SYNCHRONIZED_ SUSPENDED. 1 = HADR_SYNCHRONIZED_ SUSPENDED. When a forced failover completes, each of the secondary databases is set to HADR_SYNCHONIZED_SUSPENDED and remains in this state until the new primary replica receives an acknowledgement from that secondary database to the SUSPEND message. NULL = Unknown (no quorum)
is_database_joined	**bit**	Indicates whether the database on this availability replica has been joined to the availability group, one of: 0 = Database is not joined to the availability group on this availability replica. 1 = Database is joined to the availability group on this availability replica. NULL = unknown (The availability replica lacks quorum.)
recovery_lsn	**numeric(25,0)**	On the primary replica, the end of the transaction log before the replica writes any new log records after recovery or failover. On the primary replica, the row for a given secondary database will have the value to which the primary replica needs the secondary replica to synchronize to (that is, to revert to and reinitialize to). On secondary replicas this value is NULL. Note that each secondary replica will have either the MAX value or a lower value that the primary replica has told the secondary replica to go back to.
truncation_lsn	**numeric(25,0)**	The Always On availability groups log truncation value, which may be higher than the local truncation LSN if local log truncation is blocked (such as by a backup operation).

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Always On Availability Groups Dynamic Management Views and Functions (Transact-SQL)
Always On Availability Groups Catalog Views (Transact-SQL)
Monitor Availability Groups (Transact-SQL)
Always On Availability Groups (SQL Server)
sys.dm_hadr_database_replica_states (Transact-SQL)

sys.dm_hadr_database_replica_states (Transact-SQL)

5/3/2018 • 10 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each database that is participating in an Always On availability group for which the local instance of SQL Server is hosting an availability replica. This dynamic management view exposes state information on both the primary and secondary replicas. On a secondary replica, this view returns a row for every secondary database on the server instance. On the primary replica, this view returns a row for each primary database and an additional row for the corresponding secondary database.

> **IMPORTANT**
> Depending on the action and higher-level states, database-state information may be unavailable or out of date. Furthermore, the values have only local relevance. For example, on the primary replica, the value of the **last_hardened_lsn** column reflects the information about a given secondary database that is currently available to the primary replica, not the actual hardened LSN value that the secondary replica might have currently.

COLUMN NAME	DATA TYPE	DESCRIPTION (ON PRIMARY REPLICA)
database_id	**int**	Identifier of the database, unique within an instance of SQL Server. This is the same value as displayed in the sys.databases catalog view.
group_id	**uniqueidentifier**	Identifier of the availability group to which the database belongs.
replica_id	**uniqueidentifier**	Identifier of the availability replica within the availability group.
group_database_id	**uniqueidentifier**	Identifier of the database within the availability group. This identifier is identical on every replica to which this database is joined.
is_local	**bit**	Whether the availability database is local, one of: 0 = The database is not local to the SQL Server instance. 1 = The database is local to the server instance.
is_primary_replica	**bit**	Returns 1 if the replica is primary, or 0 if it is a secondary replica. **Applies to:** SQL Server 2014 (12.x) through SQL Server 2017.

COLUMN NAME	DATA TYPE	DESCRIPTION (ON PRIMARY REPLICA)
synchronization_state	**tinyint**	Data-movement state, one of the following values. 0 = Not synchronizing. For a primary database, indicates that the database is not ready to synchronize its transaction log with the corresponding secondary databases. For a secondary database, indicates that the database has not started log synchronization because of a connection issue, is being suspended, or is going through transition states during startup or a role switch. 1 = Synchronizing. For a primary database, indicates that the database is ready to accept a scan request from a secondary database. For a secondary database, indicates that active data movement is occurring for the database. 2 = Synchronized. A primary database shows SYNCHRONIZED in place of SYNCHRONIZING. A synchronous-commit secondary database shows synchronized when the local cache says the database is failover ready and is synchronizing. 3 = Reverting. Indicates the phase in the undo process when a secondary database is actively getting pages from the primary database. **Caution:** When a database on a secondary replica is in the REVERTING state, forcing failover to the secondary replica leaves the database in a state in which it cannot be started as a primary database. Either the database will need to reconnect as a secondary database, or you will need to apply new log records from a log backup. 4 = Initializing. Indicates the phase of undo when the transaction log required for a secondary database to catch up to the undo LSN is being shipped and hardened on a secondary replica. **Caution:** When a database on a secondary replica is in the INITIALIZING state, forcing failover to the secondary replica leaves the database in a state in which it be started as a primary database. Either the database will need to reconnect as a secondary database, or you will need to apply new log records from a log backup.

COLUMN NAME	DATA TYPE	DESCRIPTION (ON PRIMARY REPLICA)
synchronization_state_desc	**nvarchar(60)**	Description of the data-movement state, one of: NOT SYNCHRONIZING SYNCHRONIZING SYNCHRONIZED REVERTING INITIALIZING
is_commit_participant	**bit**	0 = Transaction commit is not synchronized with respect to this database. 1 = Transaction commit is synchronized with respect to this database. For a database on an asynchronous-commit availability replica, this value is always 0. For a database on a synchronous-commit availability replica, this value is accurate only on the primary database.
synchronization_health	**tinyint**	Reflects the intersection of the synchronization state of a database that is joined to the availability group on the availability replica and the availability mode of the availability replica (synchronous-commit or asynchronous-commit mode), one of the following values. 0 = Not healthy. The **synchronization_state** of the database is 0 (NOT SYNCHRONIZING). 1 = Partially healthy. A database on a synchronous-commit availability replica is considered partially healthy if **synchronization_state** is 1 (SYNCHRONIZING). 2 = Healthy. A database on an synchronous-commit availability replica is considered healthy if **synchronization_state** is 2 (SYNCHRONIZED), and a database on an asynchronous-commit availability replica is considered healthy if **synchronization_state** is 1 (SYNCHRONIZING).

COLUMN NAME	DATA TYPE	DESCRIPTION (ON PRIMARY REPLICA)
synchronization_health_desc	**nvarchar(60)**	Description of the **synchronization_health** of the availability database. NOT_HEALTHY PARTIALLY_HEALTHY HEALTHY
database_state	**tinyint**	0 = Online 1 = Restoring 2 = Recovering 3 = Recovery pending 4 = Suspect 5 = Emergency 6 = Offline **Note:** Same as **state** column in sys.databases.
database_state_desc	**nvarchar(60)**	Description of the **database_state** of the availability replica. ONLINE RESTORING RECOVERING RECOVERY_PENDING SUSPECT EMERGENCY OFFLINE **Note:** Same as **state_desc** column in sys.databases.
is_suspended	**bit**	Database state, one of: 0 = Resumed 1 = Suspended

COLUMN NAME	DATA TYPE	DESCRIPTION (ON PRIMARY REPLICA)
suspend_reason	**tinyint**	If the database is suspended, the reason for the suspended state, one of: 0 = User action 1 = Suspend from partner 2 = Redo 3 = Capture 4 = Apply 5 = Restart 6 = Undo 7 = Revalidation 8 = Error in the calculation of the secondary-replica synchronization point
suspend_reason_desc	**nvarchar(60)**	Description of the database suspended state reason, one of: SUSPEND_FROM_USER = A user manually suspended data movement SUSPEND_FROM_PARTNER = The database replica is suspended after a forced failover SUSPEND_FROM_REDO = An error occurred during the redo phase SUSPEND_FROM_APPLY = An error occurred when writing the log to file (see error log) SUSPEND_FROM_CAPTURE = An error occurred while capturing log on the primary replica SUSPEND_FROM_RESTART = The database replica was suspended before the database was restarted (see error log) SUSPEND_FROM_UNDO = An error occurred during the undo phase (see error log) SUSPEND_FROM_REVALIDATION = Log change mismatch is detected on reconnection (see error log) SUSPEND_FROM_XRF_UPDATE = Unable to find the common log point (see error log)

COLUMN NAME	DATA TYPE	DESCRIPTION (ON PRIMARY REPLICA)
recovery_lsn	**numeric(25,0)**	On the primary replica, the end of the transaction log before the primary database writes any new log records after recovery or failover. For a given secondary database, if this value is less than the current hardened LSN (last_hardened_lsn), recovery_lsn is the value to which this secondary database would need to resynchronize (that is, to revert to and reinitialize to). If this value is greater than or equal to the current hardened LSN, resynchronization would be unnecessary and would not occur. **recovery_lsn** reflects a log-block ID padded with zeroes. It is not an actual log sequence number (LSN). For information about how this value is derived, see Understanding the LSN Column Values, later in this topic.
truncation_lsn	**numeric(25,0)**	On the primary replica, for the primary database, reflects the minimum log truncation LSN across all the corresponding secondary databases. If local log truncation is blocked (such as by a backup operation), this LSN might be higher than the local truncation LSN. For a given secondary database, reflects the truncation point of that database. **truncation_lsn** reflects a log-block ID padded with zeroes. It is not an actual log sequence number.
last_sent_lsn	**numeric(25,0)**	The log block identifier that indicates the point up to which all log blocks have been sent by the primary. This is the ID of the next log block that will be sent, rather than the ID of the most recently sent log block. **last_sent_lsn** reflects a log-block ID padded with zeroes, It is not an actual log sequence number.
last_sent_time	**datetime**	Time when the last log block was sent.
last_received_lsn	**numeric(25,0)**	Log block ID identifying the point up to which all log blocks have been received by the secondary replica that hosts this secondary database. **last_received_lsn** reflects a log-block ID padded with zeroes. It is not an actual log sequence number.

COLUMN NAME	DATA TYPE	DESCRIPTION (ON PRIMARY REPLICA)
last_received_time	**datetime**	Time when the log block ID in last message received was read on the secondary replica.
last_hardened_lsn	**numeric(25,0)**	Start of the Log Block containing the log records of last hardened LSN on a secondary database. On an asynchronous-commit primary database or on a synchronous-commit database whose current policy is "delay", the value is NULL. For other synchronous-commit primary databases, **last_hardened_lsn** indicates the minimum of the hardened LSN across all the secondary databases. **Note: last_hardened_lsn** reflects a log-block ID padded with zeroes. It is not an actual log sequence number. For more information, see Understanding the LSN Column Values, later in this topic.
last_hardened_time	**datetime**	On a secondary database, time of the log-block identifier for the last hardened LSN (**last_hardened_lsn**). On a primary database, reflects the time corresponding to minimum hardened LSN.
last_redone_lsn	**numeric(25,0)**	Actual log sequence number of the last log record that was redone on the secondary database. **last_redone_lsn** is always less than **last_hardened_lsn**.
last_redone_time	**datetime**	Time when the last log record was redone on the secondary database.
log_send_queue_size	**bigint**	Amount of log records of the primary database that has not been sent to the secondary databases, in kilobytes (KB).
log_send_rate	**bigint**	Average rate at which primary replica instance sent data during last active period, in kilobytes (KB)/second.
redo_queue_size	**bigint**	Amount of log records in the log files of the secondary replica that has not yet been redone, in kilobytes (KB).
redo_rate	**bigint**	Rate at which the log records are being redone on a given secondary database, in kilobytes (KB)/second.

COLUMN NAME	DATA TYPE	DESCRIPTION (ON PRIMARY REPLICA)
filestream_send_rate	**bigint**	The rate at which the FILESTREAM files are shipped to the secondary replica, in kilobytes (KB)/second.
end_of_log_lsn	**numeric(25,0)**	Local end of log LSN. Actual LSN corresponding to the last log record in the log cache on the primary and secondary databases. On the primary replica, the secondary rows reflect the end of log LSN from the latest progress messages that the secondary replicas have sent to the primary replica. **end_of_log_lsn** reflects a log-block ID padded with zeroes. It is not an actual log sequence number. For more information, see Understanding the LSN Column Values, later in this topic.
last_commit_lsn	**Numeric(25,0)**	Actual log sequence number corresponding to the last commit record in the transaction log. On the primary database, this corresponds to the last commit record processed. Rows for secondary databases show the log sequence number that the secondary replica has sent to the primary replica. On the secondary replica, this is the last commit record that was redone.
last_commit_time	**datetime**	Time corresponding to the last commit record. On the secondary database, this time is the same as on the primary database. On the primary replica, each secondary database row displays the time that the secondary replica that hosts that secondary database has reported back to the primary replica. The difference in time between the primary-database row and a given secondary-database row represents approximately the recovery time objective (RPO), assuming that the redo process is caught up and that the progress has been reported back to the primary replica by the secondary replica.

COLUMN NAME	DATA TYPE	DESCRIPTION (ON PRIMARY REPLICA)
low_water_mark_for_ghosts	**bigint**	A monotonically increasing number for the database indicating a low water mark used by ghost cleanup on the primary database. If this number is not increasing over time, it implies that ghost cleanup might not happen. To decide which ghost rows to clean up, the primary replica uses the minimum value of this column for this database across all availability replicas (including the primary replica).
secondary_lag_seconds	**bigint**	The number of seconds that the secondary replica is behind the primary replica during synchronization. **Applies to:** SQL Server 2016 (13.x) through SQL Server 2017.

Understanding the LSN Column Values

The values of the **end_of_log_lsn**, **last_hardened_lsn**, **last_received_lsn**, **last_sent_lsn**, **recovery_lsn**, and **truncation_lsn** columns are not actual log sequence numbers (LSNs). Rather each of these values reflects a log-block ID padded with zeroes.

end_of_log_lsn, **last_hardened_lsn**, and **recovery_lsn** are flush LSNs. For example, **last_hardened_lsn** indicates the start of the next block past the blocks that are already on disk. So any LSN < the value of **last_hardened_lsn** is on disk. LSN that are >= to this value are not flushed.

Of the LSN values returned by **sys.dm_hadr_database_replica_states**, only **last_redone_lsn** is a real LSN.

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Always On Availability Groups (SQL Server)
Monitor Availability Groups (Transact-SQL)

sys.dm_hadr_instance_node_map (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

For every instance of SQL Server that hosts an availability replica that is joined to its Always On availability group, returns the name of the Windows Server Failover Clustering (WSFC) node that hosts the server instance. This dynamic management view has the following uses:

- This dynamic management view is useful for detecting an availability group with multiple availability replicas that are hosted on the same WSFC node, which is an unsupported configuration that could occur after an FCI failover if the availability group is incorrectly configured. For more information, see Failover Clustering and Always On Availability Groups (SQL Server).
- When multiple SQL Server instances are hosted on the same WSFC node, the Resource DLL uses this dynamic management view to determine the instance of SQL Server to connect to.

COLUMN NAME	DATA TYPE	DESCRIPTION
ag_resource_id	**nvarchar(256)**	Unique ID of the availability group as a resource in the WSFC cluster.
instance_name	**nvarchar(256)**	Name—*server/instance*—of a server instance that hosts a replica for the availability group.
node_name	**nvarchar(256)**	Name of the WSFC cluster node.

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Always On Availability Groups Dynamic Management Views and Functions (Transact-SQL)
Always On Availability Groups Catalog Views (Transact-SQL)
Monitor Availability Groups (Transact-SQL)
Always On Availability Groups (SQL Server)

sys.dm_hadr_name_id_map (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Shows the mapping of Always On availability groups that the current instance of SQL Server has joined to three unique IDs: an availability group ID, a WSFC resource ID, and a WSFC Group ID. The purpose of this mapping is to handle the scenario in which the WSFC resource/group is renamed.

COLUMN NAME	DATA TYPE	DESCRIPTION
ag_name	**nvarchar(256)**	Name of the availability group. This is a user-specified name that must be unique within the Windows Server Failover Cluster (WSFC) cluster.
ag_id	**uniqueidentifier**	Unique identifier (GUID) of the availability group.
ag_resource_id	**nvarchar(256)**	Unique ID of the availability group as a resource in the WSFC cluster.
ag_group_id	**nvarchar(256)**	Unique WSFC Group ID of the availability group.

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Always On Availability Groups Dynamic Management Views and Functions (Transact-SQL)
Always On Availability Groups Catalog Views (Transact-SQL)
Monitor Availability Groups (Transact-SQL)
Always On Availability Groups (SQL Server)

sys.dm_tcp_listener_states (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row containing dynamic-state information for each TCP listener.

> **NOTE**
> The availability group listener could listen to the same port as the listener of the instance of SQL Server. In this case, the listeners are listed separately, the same as for a Service Broker listener.

COLUMN NAME	DATA TYPE	DESCRIPTION
listener_id	**int**	Listener's internal ID. Is not nullable. Primary key.
ip_address	**nvarchar48**	The listener IP address that is online and currently being listening to. Either IPv4 and IPv6 is allowed. If a listener possesses both types of addresses, they are listed separately. An IPv4 wildcard, is displayed as "0.0.0.0". An IPv6 wildcard, is displayed as "::". Is not nullable.
is_ipv4	**bit**	Type of IP address 1 = IPv4 0 = IPv6
port	**int**	The port number on which the listener is listening. Is not nullable.
type	**tinyint**	Listener type, one of: 0 = Transact-SQL 1 = Service Broker 2 = Database mirroring Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
type_desc	**nvarchar(20)**	Description of the **type**, one of: TSQL SERVICE_BROKER DATABASE_MIRRORING Is not nullable.
state	**tinyint**	State of the availability group listener, one of: 1 = Online. The listener is listening and processing requests. 2 = Pending restart. the listener is offline, pending a restart. If the availability group listener is listening to the same port as the server instance, these two listeners always have the same state. Is not nullable. Note: The values in this column come from the TSD_listener object. The column does not support an offline state because when the TDS_listener is offline, it cannot be queried for state.
state_desc	**nvarchar(16)**	Description of **state**, one of: ONLINE PENDING_RESTART Is not nullable.
start_time	**datetime**	Timestamp indicating when the listener was started. Is not nullable.

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Querying the SQL Server System Catalog FAQ
Always On Availability Groups Catalog Views (Transact-SQL)
Always On Availability Groups Dynamic Management Views and Functions (Transact-SQL)

Change Data Capture - sys.dm_cdc_errors

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row for each error encountered during the change data capture log scan session.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**int**	ID of the session. 0 = the error did not occur within a log scan session.
phase_number	**int**	Number indicating the phase the session was in at the time the error occurred. For a description of each phase, see sys.dm_cdc_log_scan_sessions (Transact-SQL).
entry_time	**datetime**	The date and time the error was logged. This value corresponds to the timestamp in the SQL error log.
error_number	**int**	ID of the error message.
error_severity	**int**	Severity level of the message, between 1 and 25.
error_state	**int**	State number of the error.
error_message	**nvarchar(1024)**	Message text of the error.
start_lsn	**nvarchar(23)**	Starting LSN value of the rows being processed when the error occurred. 0 = the error did not occur within a log scan session.
begin_lsn	**nvarchar(23)**	Beginning LSN value of the transaction being processed when the error occurred. 0 = the error did not occur within a log scan session.
sequence_value	**nvarchar(23)**	LSN value of the rows being processed when the error occurred. 0 = the error did not occur within a log scan session.

Remarks

sys.dm_cdc_errors contains error information for the previous 32 sessions.

Permissions

Requires VIEW DATABASE STATE permission to query the **sys.dm_cdc_errors** dynamic management view. For more information about permissions on dynamic management views, see Dynamic Management Views and Functions (Transact-SQL).

See Also

sys.dm_cdc_log_scan_sessions (Transact-SQL)
sys.dm_repl_traninfo (Transact-SQL)

Change Data Capture - sys.dm_cdc_log_scan_sessions

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row for each log scan session in the current database. The last row returned represents the current session. You can use this view to return status information about the current log scan session, or aggregated information about all sessions since the instance of SQL Server was last started.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**int**	ID of the session. 0 = the data returned in this row is an aggregate of all sessions since the instance of SQL Server was last started.
start_time	**datetime**	Time the session began. When **session_id** = 0, the time aggregated data collection began.
end_time	**datetime**	Time the session ended. NULL = session is active. When **session_id** = 0, the time the last session ended.
duration	**bigint**	The duration (in seconds) of the session. 0 = the session does not contain change data capture transactions. When **session_id** = 0, the sum of the duration (in seconds) of all sessions with change data capture transactions.
scan_phase	**nvarchar(200)**	The current phase of the session. The following are the possible values and their descriptions: 1: Reading configuration 2: First scan, building hash table 3: Second scan 4: Second scan 5: Second scan 6: Schema versioning 7: Last scan 8: Done When **session_id** = 0, this value is always "Aggregate".

COLUMN NAME	DATA TYPE	DESCRIPTION
error_count	**int**	Number of errors encountered. When **session_id** = 0, the total number of errors in all sessions.
start_lsn	**nvarchar(23)**	Starting LSN for the session. When **session_id** = 0, the starting LSN for the last session.
current_lsn	**nvarchar(23)**	Current LSN being scanned. When **session_id** = 0, the current LSN is 0.
end_lsn	**nvarchar(23)**	Ending LSN for the session. NULL = session is active. When **session_id** = 0, the ending LSN for the last session.
tran_count	**bigint**	Number of change data capture transactions processed. This counter is populated in phase 2. When **session_id** = 0, the number of processed transactions in all sessions.
last_commit_lsn	**nvarchar(23)**	LSN of the last commit log record processed. When **session_id** = 0, the last commit log record LSN for any session.
last_commit_time	**datetime**	Time the last commit log record was processed. When **session_id** = 0, the time the last commit log record for any session.
log_record_count	**bigint**	Number of log records scanned. When **session_id** = 0, number of records scanned for all sessions.
schema_change_count	**int**	Number of data definition language (DDL) operations detected. This counter is populated in phase 6. When **session_id** = 0, the number of DDL operations processed in all sessions.

COLUMN NAME	DATA TYPE	DESCRIPTION
command_count	**bigint**	Number of commands processed. When **session_id** = 0, the number of commands processed in all sessions.
first_begin_cdc_lsn	**nvarchar(23)**	First LSN that contained change data capture transactions. When **session_id** = 0, the first LSN that contained change data capture transactions.
last_commit_cdc_lsn	**nvarchar(23)**	LSN of the last commit log record that contained change data capture transactions. When **session_id** = 0, the last commit log record LSN for any session that contained change data capture transactions
last_commit_cdc_time	**datetime**	Time the last commit log record was processed that contained change data capture transactions. When **session_id** = 0, the time the last commit log record for any session that contained change data capture transactions.
latency	**int**	The difference, in seconds, between **end_time** and **last_commit_cdc_time** in the session. This counter is populated at the end of phase 7. When **session_id** = 0, the last nonzero latency value recorded by a session.
empty_scan_count	**int**	Number of consecutive sessions that contained no change data capture transactions.
failed_sessions_count	**int**	Number of sessions that failed.

Remarks

The values in this dynamic management view are reset whenever the instance of SQL Server is started.

Permissions

Requires VIEW DATABASE STATE permission to query the **sys.dm_cdc_log_scan_sessions** dynamic management view. For more information about permissions on dynamic management views, see Dynamic Management Views and Functions (Transact-SQL).

Examples

The following example returns information for the most current session.

```
USE AdventureWorks2012;
GO
SELECT session_id, start_time, end_time, duration, scan_phase
    error_count, start_lsn, current_lsn, end_lsn, tran_count
    last_commit_lsn, last_commit_time, log_record_count, schema_change_count
    command_count, first_begin_cdc_lsn, last_commit_cdc_lsn,
    last_commit_cdc_time, latency, empty_scan_count, failed_sessions_count
FROM sys.dm_cdc_log_scan_sessions
WHERE session_id = (SELECT MAX(b.session_id) FROM sys.dm_cdc_log_scan_sessions AS b);
GO
```

See Also

sys.dm_cdc_errors (Transact-SQL)

Change Tracking - sys.dm_tran_commit_table

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Displays one row for each transaction that is committed for a table that is tracked by SQL Server change tracking. The sys.dm_tran_commit_table management view, which is provided for supportability purposes and exposes the transaction-related information that change tracking stores in the sys.syscommittab system table. The sys.syscommittab table provides an efficient persistent mapping from a database-specific transaction ID to the transaction's commit log sequence number (LSN) and commit timestamp. The data that is stored in the sys.syscommittab table and exposed in this management view is subject to cleanup according to the retention period specified when change tracking was configured.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_tran_commit_table**.

COLUMN NAME	DATA TYPE	DESCRIPTION
commit_ts	**bigint**	A monotonically increasing number that serves as a database-specific timestamp for each committed transaction.
xdes_id	**bigint**	A database-specific internal ID for the transaction.
commit_lbn	**bigint**	The number of the log block that contains the commit log record for the transaction.
commit_csn	**bigint**	The instance-specific commit sequence number for the transaction.
commit_time	**smalldatetime**	The time when the transaction was committed.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

See Also

Dynamic Management Views and Functions (Transact-SQL)
About Change Tracking (SQL Server)

Common Language Runtime Related Dynamic Management Views (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management views.

sys.dm_clr_appdomains	sys.dm_clr_loaded_assemblies
sys.dm_clr_properties	sys.dm_clr_tasks

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_clr_appdomains (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each application domain in the server. Application domain (**AppDomain**) is a construct in the Microsoft .NET Framework common language runtime (CLR) that is the unit of isolation for an application. You can use this view to understand and troubleshoot CLR integration objects that are executing in Microsoft SQL Server.

There are several types of CLR integration managed database objects. For general information about these objects, see Building Database Objects with Common Language Runtime (CLR) Integration. Whenever these objects are executed, SQL Server creates an **AppDomain** under which it can load and execute the required code. The isolation level for an **AppDomain** is one **AppDomain** per database per owner. That is, all CLR objects owned by a user are always executed in the same **AppDomain** per-database (if a user registers CLR database objects in different databases, the CLR database objects will run in different application domains). An **AppDomain** is not destroyed after the code finishes execution. Instead, it is cached in memory for future executions. This improves performance.

For more information, see Application Domains.

COLUMN NAME	DATA TYPE	DESCRIPTION
appdomain_address	**varbinary(8)**	Address of the **AppDomain**. All managed database objects owned by a user are always loaded in the same **AppDomain**. You can use this column to look up all the assemblies currently loaded in this **AppDomain** in **sys.dm_clr_loaded_assemblies**.
appdomain_id	**int**	ID of the **AppDomain**. Each **AppDomain** has a unique ID.
appdomain_name	**varchar(386)**	Name of the **AppDomain** as assigned by SQL Server.
creation_time	**datetime**	Time when the **AppDomain** was created. Because **AppDomains** are cached and reused for better performance, **creation_time** is not necessarily the time when the code was executed.
db_id	**int**	ID of the database in which this **AppDomain** was created. Code stored in two different databases cannot share one **AppDomain**.
user_id	**int**	ID of the user whose objects can execute in this **AppDomain**.

COLUMN NAME	DATA TYPE	DESCRIPTION
state	**nvarchar(128)**	A descriptor for the current state of the **AppDomain**. An AppDomain can be in different states from creation to deletion. See the Remarks section of this topic for more information.
strong_refcount	**int**	Number of strong references to this **AppDomain**. This reflects the number of currently executing batches that use this **AppDomain**. Note that execution of this view will create a **strong refcount**; even if is no code currently executing, **strong_refcount** will have a value of 1.
weak_refcount	**int**	Number of weak references to this **AppDomain**. This indicates how many objects inside the **AppDomain** are cached. When you execute a managed database object, SQL Server caches it inside the **AppDomain** for future reuse. This improves performance.
cost	**int**	Cost of the **AppDomain**. The higher the cost, the more likely this **AppDomain** is to be unloaded under memory pressure. Cost usually depends on how much memory is required to re-create this **AppDomain**.
value	**int**	Value of the **AppDomain**. The lower the value, the more likely this **AppDomain** is to be unloaded under memory pressure. Value usually depends on how many connections or batches are using this **AppDomain**.
total_processor_time_ms	**bigint**	Total processor time, in milliseconds, used by all threads while executing in the current application domain since the process started. This is equivalent to **System.AppDomain.MonitoringTotalProcessorTime**.
total_allocated_memory_kb	**bigint**	Total size, in kilobytes, of all memory allocations that have been made by the application domain since it was created, without subtracting memory that has been collected. This is equivalent to **System.AppDomain.MonitoringTotalAllocatedMemorySize**.

COLUMN NAME	DATA TYPE	DESCRIPTION
survived_memory_kb	**bigint**	Number of kilobytes that survived the last full, blocking collection and that are known to be referenced by the current application domain. This is equivalent to **System.AppDomain.MonitoringSurvivedMemorySize**.

Remarks

There is a one-to-may relationship between **dm_clr_appdomains.appdomain_address** and **dm_clr_loaded_assemblies.appdomain_address**.

The following tables list possible **state** values, their descriptions, and when they occur in the **AppDomain** lifecycle. You can use this information to follow the lifecyle of an **AppDomain** and to watch for suspicious or repetitive **AppDomain** instances unloading, without having to parse the Windows Event Log.

AppDomain Initialization

STATE	DESCRIPTION
E_APPDOMAIN_CREATING	The **AppDomain** is being created.

AppDomain Usage

STATE	DESCRIPTION
E_APPDOMAIN_SHARED	The runtime **AppDomain** is ready for use by multiple users.
E_APPDOMAIN_SINGLEUSER	The **AppDomain** is ready for use in DDL operations. These differ from E_APPDOMAIN_SHARED in that shared AppDomains are used for CLR integration executions as opposed to DDL operations. Such AppDomains are isolated from other concurrent operations.
E_APPDOMAIN_DOOMED	The **AppDomain** is scheduled to be unloaded, but there are currently threads executing in it.

AppDomain Cleanup

STATE	DESCRIPTION
E_APPDOMAIN_UNLOADING	SQL Server has requested that the CLR unload the **AppDomain**, usually because the assembly that contains the managed database objects has been altered or dropped.
E_APPDOMAIN_UNLOADED	The CLR has unloaded the **AppDomain**. This is usually the result of an escalation procedure due to **ThreadAbort**, **OutOfMemory**, or an unhandled exception in user code.
E_APPDOMAIN_ENQUEUE_DESTROY	The **AppDomain** has been unloaded in CLR and set to be destroyed by SQL Server.

STATE	DESCRIPTION
E_APPDOMAIN_DESTROY	The **AppDomain** is in the process of being destroyed by SQL Server.
E_APPDOMAIN_ZOMBIE	The **AppDomain** has been destroyed by SQL Server; however, not all of the references to the **AppDomain** have been cleaned up.

Permissions

Requires VIEW SERVER STATE permission on the database.

Examples

The following example shows how to view the details of an **AppDomain** for a given assembly:

```
select appdomain_id, creation_time, db_id, user_id, state
from sys.dm_clr_appdomains a
where appdomain_address =
(select appdomain_address
 from sys.dm_clr_loaded_assemblies
   where assembly_id = 500);
```

The following example shows how to view all assemblies in a given **AppDomain**:

```
select a.name, a.assembly_id, a.permission_set_desc, a.is_visible, a.create_date, l.load_time
from sys.dm_clr_loaded_assemblies as l
inner join sys.assemblies as a
on l.assembly_id = a.assembly_id
where l.appdomain_address =
(select appdomain_address
from sys.dm_clr_appdomains
where appdomain_id = 15);
```

See Also

sys.dm_clr_loaded_assemblies (Transact-SQL)
Common Language Runtime Related Dynamic Management Views (Transact-SQL)

sys.dm_clr_loaded_assemblies (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each managed user assembly loaded into the server address space. Use this view to understand and troubleshoot CLR integration managed database objects that are executing in Microsoft SQL Server.

Assemblies are managed code DLL files that are used to define and deploy managed database objects in SQL Server. Whenever a user executes one of these managed database objects, SQL Server and the CLR load the assembly (and its references) in which the managed database object is defined. The assembly remains loaded in SQL Server to increase performance, so that the managed database objects contained in the assembly can be called in the future with out having to reload the assembly. The assembly is not unloaded until SQL Server comes under memory pressure. For more information about assemblies and CLR integration, see CLR Hosted Environment. For more information about managed database objects, see Building Database Objects with Common Language Runtime (CLR) Integration.

COLUMN NAME	DATA TYPE	DESCRIPTION
assembly_id	**int**	ID of the loaded assembly. The **assembly_id** can be used to look up more information about the assembly in the sys.assemblies (Transact-SQL) catalog view. Note that the Transact-SQL sys.assemblies catalog shows assemblies in the current database only. The **sqs.dm_clr_loaded_assemblies** view shows all loaded assemblies on the server.
appdomain_address	**int**	Address of the application domain (**AppDomain**) in which the assembly is loaded. All the assemblies owned by a single user are always loaded in the same **AppDomain**. The **appdomain_address** can be used to lookup more information about the **AppDomain** in the sys.dm_clr_appdomains view.
load_time	**datetime**	Time when the assembly was loaded. Note that the assembly remains loaded until SQL Server is under memory pressure and unloads the **AppDomain**. You can monitor **load_time** to understand how frequently SQL Server comes under memory pressure and unloads the **AppDomain**.

Permissions

Requires VIEW SERVER STATE permission on the server.

Remarks

The **dm_clr_loaded_assemblies.appdomain_address** view has a many-to-one relationship with **dm_clr_appdomains.appdomain_address**. The **dm_clr_loaded_assemblies.assembly_id** view has a one-to-many relationship with **sys.assemblies.assembly_id**.

Examples

The following example shows how to view details of all assemblies in the current database that are currently loaded.

```
 SELECT a.name, a.assembly_id, a.permission_set_desc, a.is_visible, a.create_date, l.load_time
FROM sys.dm_clr_loaded_assemblies AS l
INNER JOIN sys.assemblies AS a
ON l.assembly_id = a.assembly_id;
```

The following example shows how to view details of the **AppDomain** in which a given assembly is loaded.

```
SELECT appdomain_id, creation_time, db_id, user_id, state
FROM sys.dm_clr_appdomains AS a
WHERE appdomain_address =
(SELECT appdomain_address
 FROM sys.dm_clr_loaded_assemblies
 WHERE assembly_id = 555);
```

See Also

Common Language Runtime Related Dynamic Management Views (Transact-SQL)

sys.dm_clr_properties (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each property related to SQL Server common language runtime (CLR) integration, including the version and state of the hosted CLR. The hosted CLR is initialized by running the CREATE ASSEMBLY, ALTER ASSEMBLY, or DROP ASSEMBLY statements, or by executing any CLR routine, type, or trigger. The **sys.dm_clr_properties** view does not specify whether execution of user CLR code has been enabled on the server. Execution of user CLR code is enabled by using the sp_configure stored procedure with the clr enabled option set to 1.

The **sys.dm_clr_properties** view contains the **name** and **value** columns. Each row in this view provides details about a property of the hosted CLR. Use this view to gather information about the hosted CLR, such as the CLR install directory, the CLR version, and the current state of the hosted CLR. This view can help you determine if the CLR integration code is not working because of problems with the CLR installation on the server computer.

COLUMN NAME	DATA TYPE	DESCRIPTION
name	**nvarchar(128)**	The name of the property.
value	**nvarchar(128)**	Value of the property.

Properties

The **directory** property indicates the directory that the .NET Framework was installed to on the server. There could be multiple installations of .NET Framework on the server computer and the value of this property identifies which installation SQL Server is using.

The **version** property indicates the version of the .NET Framework and hosted CLR on the server.

The **sys.dm_clr_properties** dynamic managed view can return six different values for the **state** property, which reflects the state of the SQL Server hosted CLR. They are:

- Mscoree is not loaded.
- Mscoree is loaded.
- Locked CLR version with mscoree.
- CLR is initialized.
- CLR initialization permanently failed.
- CLR is stopped.

  The **Mscoree is not loaded** and **Mscoree is loaded** states show the progression of the hosted CLR initialization on server startup, and are not likely to be seen.

  The **Locked CLR version with mscoree** state may be seen where the hosted CLR is not being used and, thus, it has not yet been initialized. The hosted CLR is initialized the first time a DDL statement (such as CREATE ASSEMBLY (Transact-SQL)) or a managed database object is executed.

The **CLR is initialized** state indicates that the hosted CLR was successfully initialized. Note that this does not indicate whether execution of user CLR code was enabled. If the execution of user CLR code is first enabled and then disabled using the Transact-SQL sp_configure stored procedure, the state value will still be **CLR is initialized**.

The **CLR initialization permanently failed** state indicates that hosted CLR initialization failed. Memory pressure is a likely cause, or it could also be the result of a failure in the hosting handshake between SQL Server and the CLR. Error message 6512 or 6513 will be thrown in such a case.

The **CLR is stopped state** is only seen when SQL Server is in the process of shutting down.

Remarks

The properties and values of this view might change in a future version of SQL Server due to enhancements of the CLR integration functionality.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example retrieves information about the hosted CLR:

```
SELECT name, value
FROM sys.dm_clr_properties;
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Common Language Runtime Related Dynamic Management Views (Transact-SQL)

sys.dm_clr_tasks (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for all common language runtime (CLR) tasks that are currently running. A Transact-SQL batch that contains a reference to a CLR routine creates a separate task for execution of all the managed code in that batch. Multiple statements in the batch that require managed code execution use the same CLR task. The CLR task is responsible for maintaining objects and state pertaining to managed code execution, as well as the transitions between the instance of SQL Server and the common language runtime.

COLUMN NAME	DATA TYPE	DESCRIPTION
task_address	**varbinary(8)**	Address of the CLR task.
sos_task_address	**varbinary(8)**	Address of the underlying Transact-SQL batch task.
appdomain_address	**varbinary(8)**	Address of the application domain in which this task is running.
state	**nvarchar(128)**	Current state of the task.
abort_state	**nvarchar(128)**	State the abort is currently in (if the task was canceled) There are multiple states involved while aborting tasks.
type	**nvarchar(128)**	Task type.
affinity_count	**int**	Affinity of the task.
forced_yield_count	**int**	Number of times the task was forced to yield.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Common Language Runtime Related Dynamic Management Views (Transact-SQL)

Database Related Dynamic Management Views (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section describes the following dynamic management objects in SQL Server and sometimes in SQL Database.

sys.dm_db_file_space_usage	sys.dm_db_fts_index_physical_stats
sys.dm_db_log_info	sys.dm_db_log_space_usage
sys.dm_db_log_stats	sys.dm_db_partition_stats
sys.dm_db_persisted_sku_features	sys.dm_db_session_space_usage
sys.dm_db_task_space_usage	sys.dm_db_uncontained_entities

DMV's unique to SQL Database or SQL Data Warehouse.

sys.dm_db_wait_stats (Azure SQL Database)	sys.dm_database_copies (Azure SQL Database)
sys.dm_db_resource_stats (Azure SQL Database)	sys.dm_db_objects_impacted_on_version_change (Azure SQL Database)
sys.dm_operation_status (Azure SQL Database)	

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_db_file_space_usage (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns space usage information for each file in the database.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_db_file_space_usage**.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**smallint**	Database ID.
file_id	**smallint**	File ID. file_id maps to file_id in sys.dm_io_virtual_file_stats and to fileid in sys.sysfiles.
filegroup_id	**smallint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Filegroup ID.
total_page_count	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Total number of pages in the file.
allocated_extent_page_count	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Total number of pages in the allocated extents in the file.
unallocated_extent_page_count	**bigint**	Total number of pages in the unallocated extents in the file. Unused pages in allocated extents are not included.

COLUMN NAME	DATA TYPE	DESCRIPTION
version_store_reserved_page_count	**bigint**	Total number of pages in the uniform extents allocated for the version store. Version store pages are never allocated from mixed extents. IAM pages are not included, because they are always allocated from mixed extents. PFS pages are included if they are allocated from a uniform extent. For more information, see sys.dm_tran_version_store (Transact-SQL).
user_object_reserved_page_count	**bigint**	Total number of pages allocated from uniform extents for user objects in the database. Unused pages from an allocated extent are included in the count. IAM pages are not included, because they are always allocated from mixed extents. PFS pages are included if they are allocated from a uniform extent. You can use the total_pages column in the sys.allocation_units catalog view to return the reserved page count of each allocation unit in the user object. However, note that the total_pages column includes IAM pages.
internal_object_reserved_page_count	**bigint**	Total number of pages in uniform extents allocated for internal objects in the file. Unused pages from an allocated extent are included in the count. IAM pages are not included, because they are always allocated from mixed extents. PFS pages are included if they are allocated from a uniform extent. There is no catalog view or dynamic management object that returns the page count of each internal object.
mixed_extent_page_count	**bigint**	Total number of allocated and unallocated pages in allocated mixed extents in the file. Mixed extents contain pages allocated to different objects. This count does include all the IAM pages in the file.

COLUMN NAME	DATA TYPE	DESCRIPTION
modified_extent_page_count	**bigint**	**Applies to**: SQL Server 2016 (13.x) SP2 through SQL Server 2017. Total number of pages modified in allocated extents of the file since last full database backup. The modified page count can be used to track amount of differential changes in the database since last full backup, to decide if differential backup is needed.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.
distribution_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The unique numeric id associated with the distribution.

Remarks

Page counts are always at the extent level. Therefore, page count values will always be a multiple of eight. The extents that contain Global Allocation Map (GAM) and Shared Global Allocation Map (SGAM) allocation pages are allocated uniform extents. They are not included in the previously described page counts. For more information about pages and extents, see Pages and Extents Architecture Guide.

The content of the current version store is in sys.dm_tran_version_store. Version store pages are tracked at the file level instead of the session and task level, because they are global resources. A session may generate versions, but the versions cannot be removed when the session ends. Version store cleanup must consider the longest running transaction that needs access to the particular version. The longest running transaction related to version store clean-up can be discovered by viewing the elapsed_time_seconds column in sys.dm_tran_active_snapshot_database_transactions.

Frequent changes in the mixed_extent_page_count column may indicate heavy use of SGAM pages. When this occurs, you may see many PAGELATCH_UP waits in which the wait resource is an SGAM page. For more information, see sys.dm_os_waiting_tasks (Transact-SQL), sys.dm_os_wait_stats (Transact-SQL), and sys.dm_os_latch_stats (Transact-SQL).

User Objects

The following objects are included in the user object page counters:

- User-defined tables and indexes
- System tables and indexes
- Global temporary tables and indexes
- Local temporary tables and indexes
- Table variables
- Tables returned in the table-valued functions

Internal Objects

Internal objects are only in tempdb. The following objects are included in the internal object page counters:

- Work tables for cursor or spool operations and temporary large object (LOB) storage
- Work files for operations such as a hash join
- Sort runs

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_db_file_space_usage.database_id, file_id	sys.dm_io_virtual_file_stats.database_id, file_id	One-to-one

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

Determing the Amount of Free Space in tempdb

The following query returns the total number of free pages and total free space in megabytes (MB) available in all files in **tempdb**.

```
USE tempdb;
GO
SELECT SUM(unallocated_extent_page_count) AS [free pages],
(SUM(unallocated_extent_page_count)*1.0/128) AS [free space in MB]
FROM sys.dm_db_file_space_usage;
```

Determining the Amount of Space Used by User Objects

The following query returns the total number of pages used by user objects and the total space used by user objects in tempdb.

```
USE tempdb;
GO
SELECT SUM(user_object_reserved_page_count) AS [user object pages used],
(SUM(user_object_reserved_page_count)*1.0/128) AS [user object space in MB]
FROM sys.dm_db_file_space_usage;
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)
sys.dm_db_task_space_usage (Transact-SQL)
sys.dm_db_session_space_usage (Transact-SQL)

sys.dm_db_fts_index_physical_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each full-text or semantic index in each table that has an associated full-text or semantic index.

Column name	Type	Description
object_id	int	Object ID of the table that contains the index.
fulltext_index_page_count	**bigint**	Logical size of the extraction in number of index pages.
keyphrase_index_page_count	**bigint**	Logical size of the extraction in number of index pages.
similarity_index_page_count	**bigint**	Logical size of the extraction in number of index pages.

General Remarks

For more information, see Manage and Monitor Semantic Search.

Metadata

For information about the status of semantic indexing, query the following dynamic management views:

- sys.dm_fts_index_population (Transact-SQL)
- sys.dm_fts_semantic_similarity_population (Transact-SQL)

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example shows how to query for the logical size of each full-text or semantic index in every table that has an associated full-text or semantic index:

```
SELECT * FROM sys.dm_db_fts_index_physical_stats;
GO
```

See Also

Manage and Monitor Semantic Search

sys.dm_db_log_info (Transact-SQL)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016 SP2) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns virtual log file (VLF) information of the transaction log. Note all transaction log files are combined in the table output. Each row in the output represents a VLF in the transaction log and provides information relevant to that VLF in the log.

Syntax

```
sys.dm_db_log_info ( database_id )
```

Arguments

database_id | NULL | DEFAULT
Is the ID of the database. *database_id* is **int**. Valid inputs are the ID number of a database, NULL, or DEFAULT. The default is NULL. NULL and DEFAULT are equivalent values in the context of current database.

Specify NULL to return VLF information of the current database.

The built-in function DB_ID can be specified. When using `DB_ID` without specifying a database name, the compatibility level of the current database must be 90 or greater.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	Database ID.
file_id	**smallint**	File id of the transaction log.
vlf_begin_offset	**bigint**	Offset location of the virtual log file (VLF) from the beginning of the transaction log file.
vlf_size_mb	**float**	virtual log file (VLF) size in MB, rounded to 2 decimal places.
vlf_sequence_number	**bigint**	virtual log file (VLF) sequence number in the created order. Used to uniquely identify VLFs in log file.
vlf_active	**bit**	Indicates whether virtual log file (VLF) is in use or not. 0 - VLF is not in use. 1 - VLF is active.

COLUMN NAME	DATA TYPE	DESCRIPTION
vlf_status	**int**	Status of the virtual log file (VLF). Possible values include 0 - VLF is inactive 1 - VLF is initialized but unused 2 - VLF is active.
vlf_parity	**tinyint**	Parity of virtual log file (VLF).Used internally to determine the end of log within a VLF.
vlf_first_lsn	**nvarchar(48)**	Log sequence number (LSN) of the first log record in the virtual log file (VLF).
vlf_create_lsn	**nvarchar(48)**	Log sequence number (LSN) of the log record that created the virtual log file (VLF).

Remarks

The `sys.dm_db_log_info` dynamic management function replaces the `DBCC LOGINFO` statement.

Permissions

Requires the `VIEW DATABASE STATE` permission in the database.

Examples

A. Determing databases in a SQL Server instance with high number of VLFs

The following query determines the databases with more than 100 VLFs in the log files, which can affect the database startup, restore, and recovery time.

```
SELECT [name], COUNT(l.database_id) AS 'vlf_count'
FROM sys.databases s
CROSS APPLY sys.dm_db_log_info(s.database_id) l
GROUP BY [name]
HAVING COUNT(l.database_id) > 100
```

B. Determing the position of the last `VLF` in transaction log before shrinking the log file

The following query can be used to determine the position of the last active VLF before running shrinkfile on transaction log to determine if transaction log can shrink.

```
USE AdventureWorks2016
GO

;WITH cte_vlf AS (
SELECT ROW_NUMBER() OVER(ORDER BY vlf_begin_offset) AS vlfid, DB_NAME(database_id) AS [Database Name],
vlf_sequence_number, vlf_active, vlf_begin_offset, vlf_size_mb
    FROM sys.dm_db_log_info(DEFAULT)),
cte_vlf_cnt AS (SELECT [Database Name], COUNT(vlf_sequence_number) AS vlf_count,
    (SELECT COUNT(vlf_sequence_number) FROM cte_vlf WHERE vlf_active = 0) AS vlf_count_inactive,
    (SELECT COUNT(vlf_sequence_number) FROM cte_vlf WHERE vlf_active = 1) AS vlf_count_active,
    (SELECT MIN(vlfid) FROM cte_vlf WHERE vlf_active = 1) AS ordinal_min_vlf_active,
    (SELECT MIN(vlf_sequence_number) FROM cte_vlf WHERE vlf_active = 1) AS min_vlf_active,
    (SELECT MAX(vlfid) FROM cte_vlf WHERE vlf_active = 1) AS ordinal_max_vlf_active,
    (SELECT MAX(vlf_sequence_number) FROM cte_vlf WHERE vlf_active = 1) AS max_vlf_active
    FROM cte_vlf
    GROUP BY [Database Name])
SELECT [Database Name], vlf_count, min_vlf_active, ordinal_min_vlf_active, max_vlf_active,
ordinal_max_vlf_active,
((ordinal_min_vlf_active-1)*100.00/vlf_count) AS free_log_pct_before_active_log,
((ordinal_max_vlf_active-(ordinal_min_vlf_active-1))*100.00/vlf_count) AS active_log_pct,
((vlf_count-ordinal_max_vlf_active)*100.00/vlf_count) AS free_log_pct_after_active_log
FROM cte_vlf_cnt
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)
sys.dm_db_log_space_usage (Transact-SQL)
sys.dm_db_log_stats (Transact-SQL)

sys.dm_db_log_space_usage (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns space usage information for the transaction log.

NOTE

All transaction log files are combined.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**smallint**	Database ID.
total_log_size_in_bytes	**bigint**	The size of the log
used_log_space_in_bytes	**bigint**	The occupied size of the log
used_log_space_in_percent	**real**	The occupied size of the log as a percent of the total log size
log_space_in_bytes_since_last_backup	**bigint**	The amount of space used since the last log backup **Applies to:** SQL Server 2014 (12.x) through SQL Server 2017, SQL Database.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.

On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

A. Determine the Amount of Free Log Space in tempdb

The following query returns the total free log space in megabytes (MB) available in tempdb.

```
USE tempdb;
GO

SELECT (total_log_size_in_bytes - used_log_space_in_bytes)*1.0/1024/1024 AS [free log space in MB]
FROM sys.dm_db_log_space_usage;
```

See Also

Dynamic Management Views and Functions (Transact-SQL)

Database Related Dynamic Management Views (Transact-SQL)

sys.dm_db_file_space_usage

sys.dm_db_task_space_usage (Transact-SQL)
sys.dm_db_session_space_usage (Transact-SQL)
sys.dm_db_log_info (Transact-SQL)
sys.dm_db_log_stats (Transact-SQL)

sys.dm_db_log_stats (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016 SP2) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns summary level attributes and information on transaction log files of databases. Use this information for monitoring and diagnostics of transaction log health.

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_db_log_stats ( database_id )
```

Arguments

database_id | NULL | **DEFAULT**

Is the ID of the database. `database_id` is `int`. Valid inputs are the ID number of a database, `NULL`, or `DEFAULT`. The default is `NULL`. `NULL` and `DEFAULT` are equivalent values in the context of current database.
The built-in function DB_ID can be specified. When using `DB_ID` without specifying a database name, the compatibility level of the current database must be 90 or greater.

Tables Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	Database ID
recovery_model	**nvarchar(60)**	Recovery model of the database. Possible values include: SIMPLE BULK_LOGGED FULL
log_min_lsn	**nvarchar(24)**	Current start log sequence number (LSN) in the transaction log.
log_end_lsn	**nvarchar(24)**	log sequence number (LSN) of the last log record in the transaction log.
current_vlf_sequence_number	**bigint**	Current virtual log file (VLF) sequence number at the time of execution.
current_vlf_size_mb	**float**	Current virtual log file (VLF) size in MB.
total_vlf_count	**bigint**	Total number of virtual log files (VLFs) in the transaction log.

COLUMN NAME	DATA TYPE	DESCRIPTION
total_log_size_mb	**float**	Total transaction log size in MB.
active_vlf_count	**bigint**	Total number of active virtual log files (VLFs) in the transaction log.
active_log_size_mb	**float**	Total active transaction log size in MB.
log_truncation_holdup_reason	**nvarchar(60)**	Log truncation holdup reason. The value is same as `log_reuse_wait_desc` column of `sys.databases`. (For more detailed explanations of these values, see The Transaction Log). Possible values include: NOTHING CHECKPOINT LOG_BACKUP ACTIVE_BACKUP_OR_RESTORE ACTIVE_TRANSACTION DATABASE_MIRRORING REPLICATION DATABASE_SNAPSHOT_CREATION LOG_SCAN AVAILABILITY_REPLICA OLDEST_PAGE XTP_CHECKPOINT OTHER TRANSIENT
log_backup_time	**datetime**	Last transaction log backup time.
log_backup_lsn	**nvarchar(24)**	Last transaction log backup log sequence number (LSN).
log_since_last_log_backup_mb	**float**	Log size in MB since last transaction log backup log sequence number (LSN).
log_checkpoint_lsn	**nvarchar(24)**	Last checkpoint log sequence number (LSN).
log_since_last_checkpoint_mb	**float**	Log size in MB since last checkpoint log sequence number (LSN).
log_recovery_lsn	**nvarchar(24)**	Recovery log sequence number (LSN) of the database. If `log_recovery_lsn` occurs before the checkpoint LSN, `log_recovery_lsn` is the oldest active transaction LSN, otherwise `log_recovery_lsn` is the checkpoint LSN.
log_recovery_size_mb	**float**	Log size in MB since log recovery log sequence number (LSN).
recovery_vlf_count	**bigint**	Total number of virtual log files (VLFs) to be recovered, if there was failover or server restart.

Permissions

Requires the `VIEW DATABASE STATE` permission in the database.

Examples

A. Determining databases in a SQL Server instance with high number of VLFs

The following query returns the databases with more than 100 VLFs in the log files. Large numbers of VLFs can affect the database startup, restore, and recovery time.

```
SELECT name AS 'Database Name', total_vlf_count AS 'VLF count'
FROM sys.databases AS s
CROSS APPLY sys.dm_db_log_stats(s.database_id)
WHERE total_vlf_count  > 100;
```

B. Determining databases in a SQL Server instance with transaction log backups older than 4 hours

The following query determines the last log backup times for the databases in the instance.

```
SELECT name AS 'Database Name', log_backup_time AS 'last log backup time'
FROM sys.databases AS s
CROSS APPLY sys.dm_db_log_stats(s.database_id);
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)
sys.dm_db_log_space_usage (Transact-SQL)
sys.dm_db_log_info (Transact-SQL)

sys.dm_db_objects_impacted_on_version_change (Azure SQL Database)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This database-scoped system view is designed to provide an early warning system to determine objects that will be impacted by a major release upgrade in Azure SQL Database. You can use the view either before or after the upgrade to get a full enumeration of impacted objects. You will need to query this view in each database to get a full accounting across the entire server.

COLUMN NAME	DATA TYPE	DESCRIPTION
class	**int** NOT NULL	The class of the object which will be impacted: **1** = constraint **7** = Indexes and heaps
class_desc	**nvarchar(60)** NOT NULL	Description of the class: **OBJECT_OR_COLUMN** **INDEX**
major_id	**int** NOT NULL	object id of the constraint, or object id of table that contains index or heap.
minor_id	**int** NULL	**NULL** for constraints Index_id for indexes and heaps
dependency	**nvarchar(60)** NOT NULL	Description of dependency that is causing a constraint or index to be impacted. The same value is also used for warnings generated during upgrade. Examples: **space** (for intrinsic) **geometry** (for system UDT) **geography::Parse** (for system UDT method)

Permissions

Requires the VIEW DATABASE STATE permission.

Example

The following example shows a query on **sys.dm_db_objects_impacted_on_version_change** to find the objects impacted by an upgrade to the next major server version

```
SELECT * FROM sys.dm_db_objects_disabled_on_version_change;
GO
```

```
class  class_desc        major_id    minor_id    dependency
------ ----------------- ----------- ----------- ----------
1      OBJECT_OR_COLUMN  181575685   NULL        geometry
7      INDEX             37575172    1           geometry
7      INDEX             2121058592  1           geometry
1      OBJECT_OR_COLUMN  101575400   NULL        geometry
```

Remarks

How to Update Impacted Objects

The following ordered steps describe the corrective action to take after the upcoming June service release upgrade.

ORDER	IMPACTED OBJECT	CORRECTIVE ACTION
1	**Indexes**	Rebuild any index identified by **sys.dm_db_objects_impacted_on_version_change** For example: `ALTER INDEX ALL ON <table> REBUILD` or `ALTER TABLE <table> REBUILD`
2	**Object**	All constraints identified by **sys.dm_db_objects_impacted_on_version_change** must be revalidated after the geometry and geography data in the underlying table is recomputed. For constraints, revalidate using ALTER TABLE. For example: `ALTER TABLE <tab> WITH CHECK CHECK CONSTRAINT <constraint name>` or `ALTER TABLE <tab> WITH CHECK CONSTRAINT ALL`

sys.dm_db_partition_stats (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns page and row-count information for every partition in the current database.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_db_partition_stats**.

COLUMN NAME	DATA TYPE	DESCRIPTION
partition_id	**bigint**	ID of the partition. This is unique within a database. This is the same value as the **partition_id** in the **sys.partitions** catalog view
object_id	**int**	Object ID of the table or indexed view that the partition is part of.
index_id	**int**	ID of the heap or index the partition is part of. 0 = Heap 1 = Clustered index. > 1 = Nonclustered index
partition_number	**int**	1-based partition number within the index or heap.
in_row_data_page_count	**bigint**	Number of pages in use for storing in-row data in this partition. If the partition is part of a heap, the value is the number of data pages in the heap. If the partition is part of an index, the value is the number of pages in the leaf level. (Nonleaf pages in the B-tree are not included in the count.) IAM (Index Allocation Map) pages are not included in either case. Always 0 for an xVelocity memory optimized columnstore index.
in_row_used_page_count	**bigint**	Total number of pages in use to store and manage the in-row data in this partition. This count includes nonleaf B-tree pages, IAM pages, and all pages included in the **in_row_data_page_count** column. Always 0 for a columnstore index.

COLUMN NAME	DATA TYPE	DESCRIPTION
in_row_reserved_page_count	**bigint**	Total number of pages reserved for storing and managing in-row data in this partition, regardless of whether the pages are in use or not. Always 0 for a columnstore index.
lob_used_page_count	**bigint**	Number of pages in use for storing and managing out-of-row **text**, **ntext**, **image**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, and **xml** columns within the partition. IAM pages are included. Total number of LOBs used to store and manage columnstore index in the partition.
lob_reserved_page_count	**bigint**	Total number of pages reserved for storing and managing out-of-row **text**, **ntext**, **image**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, and **xml** columns within the partition, regardless of whether the pages are in use or not. IAM pages are included. Total number of LOBs reserved for storing and managing a columnstore index in the partition.
row_overflow_used_page_count	**bigint**	Number of pages in use for storing and managing row-overflow **varchar**, **nvarchar**, **varbinary**, and **sql_variant** columns within the partition. IAM pages are included. Always 0 for a columnstore index.
row_overflow_reserved_page_count	**bigint**	Total number of pages reserved for storing and managing row-overflow **varchar**, **nvarchar**, **varbinary**, and **sql_variant** columns within the partition, regardless of whether the pages are in use or not. IAM pages are included. Always 0 for a columnstore index.
used_page_count	**bigint**	Total number of pages used for the partition. Computed as **in_row_used_page_count** + **lob_used_page_count** + **row_overflow_used_page_count**.
reserved_page_count	**bigint**	Total number of pages reserved for the partition. Computed as **in_row_reserved_page_count** + **lob_reserved_page_count** + **row_overflow_reserved_page_count**.

COLUMN NAME	DATA TYPE	DESCRIPTION
row_count	**bigint**	The approximate number of rows in the partition.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.
distribution_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The unique numeric id associated with the distribution.

Remarks

sys.dm_db_partition_stats displays information about the space used to store and manage in-row data LOB data, and row-overflow data for all partitions in a database. One row is displayed per partition.

The counts on which the output is based are cached in memory or stored on disk in various system tables.

In-row data, LOB data, and row-overflow data represent the three allocation units that make up a partition. The sys.allocation_units catalog view can be queried for metadata about each allocation unit in the database.

If a heap or index is not partitioned, it is made up of one partition (with partition number = 1); therefore, only one row is returned for that heap or index. The sys.partitions catalog view can be queried for metadata about each partition of all the tables and indexes in a database.

The total count for an individual table or an index can be obtained by adding the counts for all relevant partitions.

Permissions

Requires VIEW DATABASE STATE permission to query the **sys.dm_db_partition_stats** dynamic management view. For more information about permissions on dynamic management views, see Dynamic Management Views and Functions (Transact-SQL).

Examples

A. Returning all counts for all partitions of all indexes and heaps in a database

The following example shows all counts for all partitions of all indexes and heaps in the AdventureWorks2012 database.

```
USE AdventureWorks2012;
GO
SELECT * FROM sys.dm_db_partition_stats;
GO
```

B. Returning all counts for all partitions of a table and its indexes

The following example shows all counts for all partitions of the `HumanResources.Employee` table and its indexes.

```
USE AdventureWorks2012;
GO
SELECT * FROM sys.dm_db_partition_stats
WHERE object_id = OBJECT_ID('HumanResources.Employee');
GO
```

C. Returning total used pages and total number of rows for a heap or clustered index

The following example returns total used pages and total number of rows for the heap or clustered index of the `HumanResources.Employee` table. Because the `Employee` table is not partitioned by default, note the sum includes only one partition.

```
USE AdventureWorks2012;
GO
SELECT SUM(used_page_count) AS total_number_of_used_pages,
    SUM (row_count) AS total_number_of_rows
FROM sys.dm_db_partition_stats
WHERE object_id=OBJECT_ID('HumanResources.Employee')    AND (index_id=0 or index_id=1);
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_db_persisted_sku_features (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Some features of the SQL Server Database Engine change the way that Database Engine stores information in the database files. These features are restricted to specific editions of SQL Server. A database that contains these features cannot be moved to an edition of SQL Server that does not support them. Use the sys.dm_db_persisted_sku_features dynamic management view to list edition-specific features that are enabled in the current database.

Applies to: SQL Server (SQL Server 2008 through SQL Server 2017).

COLUMN NAME	DATA TYPE	DESCRIPTION
feature_name	**sysname**	External name of the feature that is enabled in the database but not supported on the all the editions of SQL Server. This feature must be removed before the database can be migrated to all available editions of SQL Server.
feature_id	**int**	Feature ID that is associated with the feature. Identified for informational purposes only. Not supported. Future compatibility is not guaranteed..

Permissions

Requires VIEW DATABASE STATE permission on the database.

Remarks

If no features that may be restricted by a specific edition are used by the database, the view returns no rows.

sys.dm_db_persisted_sku_features may list the following database-changing features as restricted to specific SQL Server editions:

- **ChangeCapture**: Indicates that a database has change data capture enabled. To remove change data capture, use the sys.sp_cdc_disable_db stored procedure. For more information, see About Change Data Capture (SQL Server).
- **ColumnStoreIndex**: Indicates that at least one table has a columnstore index. To enable a database to be moved to an edition of SQL Server that does not support this feature, use the DROP INDEX or ALTER INDEX statement to remove the columnstore index. For more information, see Columnstore indexes.

  Applies to: SQL Server (SQL Server 2012 (11.x) through SQL Server 2017).
- **Compression**: Indicates that at least one table or index uses data compression or the vardecimal storage format. To enable a database to be moved to an edition of SQL Server that does not support this feature, use the ALTER TABLE or ALTER INDEX statement to remove data compression. To remove vardecimal storage format, use the sp_tableoption statement. For more information, see Data Compression.

- **MultipleFSContainers**: Indicates that the database uses multiple FILESTREAM containers. The database has a FILESTREAM filegroup with multiple containers (files). For more information, see FILESTREAM (SQL Server).
- **InMemoryOLTP**: Indicates that the database uses In-Memory OLTP. The database has a MEMORY_OPTIMIZED_DATA filegroup. For more information, see In-Memory OLTP (In-Memory Optimization).

  Applies to: SQL Server (SQL Server 2014 (12.x) through SQL Server 2017).
- **Partitioning.** Indicates that the database contains partitioned tables, partitioned indexes, partition schemes, or partition functions. To enable a database to be moved to an edition of SQL Server other than Enterprise or Developer, it is insufficient to modify the table to be on a single partition. You must remove the partitioned table. If the table contains data, use SWITCH PARTITION to convert each partition into a nonpartitioned table. Then delete the partitioned table, the partition scheme, and the partition function.
- **TransparentDataEncryption.** Indicates that a database is encrypted by using transparent data encryption. To remove transparent data encryption, use the ALTER DATABASE statement. For more information, see Transparent Data Encryption (TDE).

> **NOTE**
>
> Starting with SQL Server 2016 (13.x) Service Pack 1, these features are available across multiple SQL Server Editions, and not limited to Enterprise or Developer Editions only.

To determine whether a database uses any features that are restricted to specific editions, execute the following statement in the database:

```
SELECT feature_name FROM sys.dm_db_persisted_sku_features;
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)
Editions and supported features of SQL Server 2016
Editions and supported features of SQL Server 2017

sys.dm_db_session_space_usage (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the number of pages allocated and deallocated by each session for the database.

> **NOTE**
> This view is applicable only to the tempdb database.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_db_session_space_usage**.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**smallint**	Session ID. **session_id** maps to **session_id** in sys.dm_exec_sessions.
database_id	**smallint**	Database ID.
user_objects_alloc_page_count	**bigint**	Number of pages reserved or allocated for user objects by this session.
user_objects_dealloc_page_count	**bigint**	Number of pages deallocated and no longer reserved for user objects by this session.
internal_objects_alloc_page_count	**bigint**	Number of pages reserved or allocated for internal objects by this session.
internal_objects_dealloc_page_count	**bigint**	Number of pages deallocated and no longer reserved for internal objects by this session.
user_objects_deferred_dealloc_page_count	**bigint**	Number of pages which have been marked for deferred deallocation. **Note:** Introduced in service packs for SQL Server 2012 (11.x) and SQL Server 2014 (12.x).
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

IAM pages are not included in any of the allocation or deallocation counts reported by this view.

Page counters are initialized to zero (0) at the start of a session. The counters track the total number of pages that have been allocated or deallocated for tasks that are already completed in the session. The counters are updated only when a task ends; they do not reflect running tasks.

A session can have multiple requests active at the same time. A request can start multiple threads, tasks, if it is a parallel query.

For more information about the sessions, requests, and tasks, see sys.dm_exec_sessions (Transact-SQL), sys.dm_exec_requests (Transact-SQL), and sys.dm_os_tasks (Transact-SQL).

User Objects

The following objects are included in the user object page counters:

- User-defined tables and indexes
- System tables and indexes
- Global temporary tables and indexes
- Local temporary tables and indexes
- Table variables
- Tables returned in the table-valued functions

Internal Objects

Internal objects are only in **tempdb**. The following objects are included in the internal object page counters:

- Work tables for cursor or spool operations and temporary large object (LOB) storage
- Work files for operations such as a hash join
- Sort runs

Physical Joins

Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_db_session_space_usage.session_id	dm_exec_sessions.session_id	One-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)
sys.dm_exec_sessions (Transact-SQL)
sys.dm_exec_requests (Transact-SQL)
sys.dm_os_tasks (Transact-SQL)
sys.dm_db_task_space_usage (Transact-SQL)
sys.dm_db_file_space_usage (Transact-SQL)

sys.dm_db_task_space_usage (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns page allocation and deallocation activity by task for the database.

> **NOTE**
>
> This view is applicable only to the tempdb database.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_db_task_space_usage**.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**smallint**	Session ID.
request_id	**int**	Request ID within the session. A request is also called a batch and may contain one or more queries. A session may have multiple requests active at the same time. Each query in the request may start multiple threads (tasks), if a parallel execution plan is used.
exec_context_id	**int**	Execution context ID of the task. For more information, see sys.dm_os_tasks (Transact-SQL).
database_id	**smallint**	Database ID.
user_objects_alloc_page_count	**bigint**	Number of pages reserved or allocated for user objects by this task.
user_objects_dealloc_page_count	**bigint**	Number of pages deallocated and no longer reserved for user objects by this task.
internal_objects_alloc_page_count	**bigint**	Number of pages reserved or allocated for internal objects by this task.
internal_objects_dealloc_page_count	**bigint**	Number of pages deallocated and no longer reserved for internal objects by this task.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

IAM pages are not included in any of the page counts reported by this view.

Page counters are initialized to zero (0) at the start of a request. These values are aggregated at the session level when the request is completed. For more information, see sys.dm_db_session_space_usage (Transact-SQL).

Work table caching, temporary table caching, and deferred drop operations affect the number of pages allocated and deallocated in a specified task.

User Objects

The following objects are included in the user object page counters:

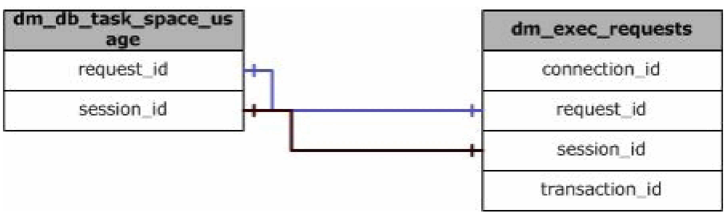
- User-defined tables and indexes
- System tables and indexes
- Global temporary tables and indexes
- Local temporary tables and indexes
- Table variables
- Tables returned in the table-valued functions

Internal Objects

Internal objects are only in **tempdb**. The following objects are included in the internal object page counters:

- Work tables for cursor or spool operations and temporary large object (LOB) storage
- Work files for operations such as a hash join
- Sort runs

Physical Joins



Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_db_task_space_usage.request_id	dm_exec_requests.request_id	One-to-one
dm_db_task_space_usage.session_id	dm_exec_requests.session_id	One-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)
sys.dm_exec_sessions (Transact-SQL)
sys.dm_exec_requests (Transact-SQL)
sys.dm_os_tasks (Transact-SQL)
sys.dm_db_session_space_usage (Transact-SQL)
sys.dm_db_file_space_usage (Transact-SQL)

sys.dm_db_uncontained_entities (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Shows any uncontained objects used in the database. Uncontained objects are objects that cross the database boundary in a contained database. This view is accessible from both a contained database and a non-contained database. If sys.dm_db_uncontained_entities is empty, your database does not use any uncontained entities.

If a module crosses the database boundary more than once, only the first discovered crossing is reported.

Column name	Type	Description
class	**int**	1 = Object or column (includes modules, XPs, views, synonyms, and tables). 4 = Database Principal 5 = Assembly 6 = Type 7 = Index (Full-text Index) 12 = Database DDL Trigger 19 = Route 30 = Audit Specification
class_desc	**nvarchar(120)**	Description of class of the entity. One of the following to match the class: **OBJECT_OR_COLUMN** **DATABASE_PRINCIPAL** **ASSEMBLY** **TYPE** **INDEX** **DATABASE_DDL_TRIGGER** **ROUTE** **AUDIT_SPECIFICATION**

major_id	**int**	ID of the entity. If *class* = 1, then object_id If *class* = 4, then sys.database_principals.principal_id. If *class* = 5, then sys.assemblies.assembly_id. If *class* = 6, then sys.types.user_type_id. If *class* = 7, then sys.indexes.index_id. If *class* = 12, then sys.triggers.object_id. If *class* = 19, then sys.routes.route_id. If *class* = 30, then sys. database_audit_specifications.databse_specification_id.
statement_line_number	**int**	If the class is a module, returns the line number on which the uncontained use is located. Otherwise the value is null.
statement_ offset_begin	**int**	If the class is a module, indicates, in bytes, beginning with 0, the starting position where uncontained use begins. Otherwise the return value is null.
statement_ offset_end	**int**	If the class is a module, indicates, in bytes, starting with 0, the ending position of the uncontained use. A value of -1 indicates the end of the module. Otherwise the return value is null.
statement_type	**nvarchar(512)**	The type of statement.
feature_ name	**nvarchar(256)**	Returns the external name of the object.
feature_type_name	**nvarchar(256)**	Returns the type of feature.

Remarks

sys.dm_db_uncontained_entities shows those entities which can potentially cross the database boundary. It will return any user entities that have the potential to use objects outside of the database.

The following feature types are reported.

- Unknown containment behavior (dynamic SQL or deferred name resolution)
- DBCC command
- System stored procedure
- System scalar function
- System table valued function

- System built-in function

Security

Permissions

sys.dm_db_uncontained_entities only returns objects for which the user has some type of permission. To fully evaluate the containment of the database this function should be used by a high privileged user such as a member of the **sysadmin** fixed server role or the **db_owner** role.

Examples

The following example creates a procedure named P1, and then queries `sys.dm_db_uncontained_entities`. The query reports that P1 uses **sys.endpoints** which is outside of the database.

```
CREATE DATABASE Test;
GO

USE Test;
GO
CREATE PROC P1
AS
SELECT * FROM sys.endpoints ;
GO
SELECT SO.name, UE.* FROM sys.dm_db_uncontained_entities AS UE
LEFT JOIN sys.objects AS SO
    ON UE.major_id = SO.object_id;
```

See Also

Contained Databases

sys.dm_db_wait_stats (Azure SQL Database)

5/4/2018 • 36 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about all the waits encountered by threads that executed during operation. You can use this aggregated view to diagnose performance issues with Azure SQL Database and also with specific queries and batches.

Specific types of wait times during query execution can indicate bottlenecks or stall points within the query. Similarly, high wait times, or wait counts server wide can indicate bottlenecks or hot spots in interaction query interactions within the server instance. For example, lock waits indicate data contention by queries; page IO latch waits indicate slow IO response times; page latch update waits indicate incorrect file layout.

COLUMN NAME	DATA TYPE	DESCRIPTION
wait_type	**nvarchar(60)**	Name of the wait type. For more information, see Types of Waits, later in this topic.
waiting_tasks_count	**bigint**	Number of waits on this wait type. This counter is incremented at the start of each wait.
wait_time_ms	**bigint**	Total wait time for this wait type in milliseconds. This time is inclusive of signal_wait_time_ms.
max_wait_time_ms	**bigint**	Maximum wait time on this wait type.
signal_wait_time_ms	**bigint**	Difference between the time that the waiting thread was signaled and when it started running.

Remarks

- This dynamic management view displays data only for the current database.
- This dynamic management view shows the time for waits that have completed. It does not show current waits.
- Counters are reset to zero any time the database is moved or taken offline.
- A SQL Server worker thread is not considered to be waiting if any of the following is true:
  - A resource becomes available.
  - A queue is nonempty.
  - An external process finishes.
- These statistics are not persisted across SQL Database failover events, and all data are cumulative since the last time the statistics were reset.

Permissions

Requires VIEW DATABASE STATE permission on the server.

Types of Waits

Resource waits
Resource waits occur when a worker requests access to a resource that is not available because the resource is being used by some other worker or is not yet available. Examples of resource waits are locks, latches, network and disk I/O waits. Lock and latch waits are waits on synchronization objects.

Queue waits
Queue waits occur when a worker is idle, waiting for work to be assigned. Queue waits are most typically seen with system background tasks such as the deadlock monitor and deleted record cleanup tasks. These tasks will wait for work requests to be placed into a work queue. Queue waits may also periodically become active even if no new packets have been put on the queue.

External waits
External waits occur when a SQL Server worker is waiting for an external event, such as an extended stored procedure call or a linked server query, to finish. When you diagnose blocking issues, remember that external waits do not always imply that the worker is idle, because the worker may actively be running some external code.

Although the thread is no longer waiting, the thread does not have to start running immediately. This is because such a thread is first put on the queue of runnable workers and must wait for a quantum to run on the scheduler.

In SQL Server the wait-time counters are **bigint** values and therefore are not as prone to counter rollover as the equivalent counters in earlier versions of SQL Server.

The following table lists the wait types encountered by tasks.

WAIT TYPE	DESCRIPTION
ABR	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
ASSEMBLY_LOAD	Occurs during exclusive access to assembly loading.
ASYNC_DISKPOOL_LOCK	Occurs when there is an attempt to synchronize parallel threads that are performing tasks such as creating or initializing a file.
ASYNC_IO_COMPLETION	Occurs when a task is waiting for I/Os to finish.
ASYNC_NETWORK_IO	Occurs on network writes when the task is blocked behind the network. Verify that the client is processing data from the server.
AUDIT_GROUPCACHE_LOCK	Occurs when there is a wait on a lock that controls access to a special cache. The cache contains information about which audits are being used to audit each audit action group.
AUDIT_LOGINCACHE_LOCK	Occurs when there is a wait on a lock that controls access to a special cache. The cache contains information about which audits are being used to audit login audit action groups.

WAIT TYPE	DESCRIPTION
AUDIT_ON_DEMAND_TARGET_LOCK	Occurs when there is a wait on a lock that is used to ensure single initialization of audit related Extended Event targets.
AUDIT_XE_SESSION_MGR	Occurs when there is a wait on a lock that is used to synchronize the starting and stopping of audit related Extended Events sessions.
BACKUP	Occurs when a task is blocked as part of backup processing.
BACKUP_OPERATOR	Occurs when a task is waiting for a tape mount. To view the tape status, query sys.dm_io_backup_tapes. If a mount operation is not pending, this wait type may indicate a hardware problem with the tape drive.
BACKUPBUFFER	Occurs when a backup task is waiting for data, or is waiting for a buffer in which to store data. This type is not typical, except when a task is waiting for a tape mount.
BACKUPIO	Occurs when a backup task is waiting for data, or is waiting for a buffer in which to store data. This type is not typical, except when a task is waiting for a tape mount.
BACKUPTHREAD	Occurs when a task is waiting for a backup task to finish. Wait times may be long, from several minutes to several hours. If the task that is being waited on is in an I/O process, this type does not indicate a problem.
BAD_PAGE_PROCESS	Occurs when the background suspect page logger is trying to avoid running more than every five seconds. Excessive suspect pages cause the logger to run frequently.
BROKER_CONNECTION_RECEIVE_TASK	Occurs when waiting for access to receive a message on a connection endpoint. Receive access to the endpoint is serialized.
BROKER_ENDPOINT_STATE_MUTEX	Occurs when there is contention to access the state of a Service Broker connection endpoint. Access to the state for changes is serialized.
BROKER_EVENTHANDLER	Occurs when a task is waiting in the primary event handler of the Service Broker. This should occur very briefly.
BROKER_INIT	Occurs when initializing Service Broker in each active database. This should occur infrequently.
BROKER_MASTERSTART	Occurs when a task is waiting for the primary event handler of the Service Broker to start. This should occur very briefly.
BROKER_RECEIVE_WAITFOR	Occurs when the RECEIVE WAITFOR is waiting. This is typical if no messages are ready to be received.
BROKER_REGISTERALLENDPOINTS	Occurs during the initialization of a Service Broker connection endpoint. This should occur very briefly.

WAIT TYPE	DESCRIPTION
BROKER_SERVICE	Occurs when the Service Broker destination list that is associated with a target service is updated or re-prioritized.
BROKER_SHUTDOWN	Occurs when there is a planned shutdown of Service Broker. This should occur very briefly, if at all.
BROKER_TASK_STOP	Occurs when the Service Broker queue task handler tries to shut down the task. The state check is serialized and must be in a running state beforehand.
BROKER_TO_FLUSH	Occurs when the Service Broker lazy flusher flushes the in-memory transmission objects to a work table.
BROKER_TRANSMITTER	Occurs when the Service Broker transmitter is waiting for work.
BUILTIN_HASHKEY_MUTEX	May occur after startup of instance, while internal data structures are initializing. Will not recur once data structures have initialized.
CHECK_PRINT_RECORD	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
CHECKPOINT_QUEUE	Occurs while the checkpoint task is waiting for the next checkpoint request.
CHKPT	Occurs at server startup to tell the checkpoint thread that it can start.
CLEAR_DB	Occurs during operations that change the state of a database, such as opening or closing a database.
CLR_AUTO_EVENT	Occurs when a task is currently performing common language runtime (CLR) execution and is waiting for a particular autoevent to be initiated. Long waits are typical, and do not indicate a problem.
CLR_CRST	Occurs when a task is currently performing CLR execution and is waiting to enter a critical section of the task that is currently being used by another task.
CLR_JOIN	Occurs when a task is currently performing CLR execution and waiting for another task to end. This wait state occurs when there is a join between tasks.
CLR_MANUAL_EVENT	Occurs when a task is currently performing CLR execution and is waiting for a specific manual event to be initiated.
CLR_MEMORY_SPY	Occurs during a wait on lock acquisition for a data structure that is used to record all virtual memory allocations that come from CLR. The data structure is locked to maintain its integrity if there is parallel access.
CLR_MONITOR	Occurs when a task is currently performing CLR execution and is waiting to obtain a lock on the monitor.

WAIT TYPE	DESCRIPTION
CLR_RWLOCK_READER	Occurs when a task is currently performing CLR execution and is waiting for a reader lock.
CLR_RWLOCK_WRITER	Occurs when a task is currently performing CLR execution and is waiting for a writer lock.
CLR_SEMAPHORE	Occurs when a task is currently performing CLR execution and is waiting for a semaphore.
CLR_TASK_START	Occurs while waiting for a CLR task to complete startup.
CLRHOST_STATE_ACCESS	Occurs where there is a wait to acquire exclusive access to the CLR-hosting data structures. This wait type occurs while setting up or tearing down the CLR runtime.
CMEMTHREAD	Occurs when a task is waiting on a thread-safe memory object. The wait time might increase when there is contention caused by multiple tasks trying to allocate memory from the same memory object.
CXPACKET	Occurs when trying to synchronize the query processor exchange iterator. You may consider lowering the degree of parallelism if contention on this wait type becomes a problem.
CXROWSET_SYNC	Occurs during a parallel range scan.
DAC_INIT	Occurs while the dedicated administrator connection is initializing.
DBMIRROR_DBM_EVENT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
DBMIRROR_DBM_MUTEX	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
DBMIRROR_EVENTS_QUEUE	Occurs when database mirroring waits for events to process.
DBMIRROR_SEND	Occurs when a task is waiting for a communications backlog at the network layer to clear to be able to send messages. Indicates that the communications layer is starting to become overloaded and affect the database mirroring data throughput.
DBMIRROR_WORKER_QUEUE	Indicates that the database mirroring worker task is waiting for more work.
DBMIRRORING_CMD	Occurs when a task is waiting for log records to be flushed to disk. This wait state is expected to be held for long periods of time.

WAIT TYPE	DESCRIPTION
DEADLOCK_ENUM_MUTEX	Occurs when the deadlock monitor and sys.dm_os_waiting_tasks try to make sure that SQL Server is not running multiple deadlock searches at the same time.
DEADLOCK_TASK_SEARCH	Large waiting time on this resource indicates that the server is executing queries on top of sys.dm_os_waiting_tasks, and these queries are blocking deadlock monitor from running deadlock search. This wait type is used by deadlock monitor only. Queries on top of sys.dm_os_waiting_tasks use DEADLOCK_ENUM_MUTEX.
DEBUG	Occurs during Transact-SQL and CLR debugging for internal synchronization.
DISABLE_VERSIONING	Occurs when SQL Server polls the version transaction manager to see whether the timestamp of the earliest active transaction is later than the timestamp of when the state started changing. If this is this case, all the snapshot transactions that were started before the ALTER DATABASE statement was run have finished. This wait state is used when SQL Server disables versioning by using the ALTER DATABASE statement.
DISKIO_SUSPEND	Occurs when a task is waiting to access a file when an external backup is active. This is reported for each waiting user process. A count larger than five per user process may indicate that the external backup is taking too much time to finish.
DISPATCHER_QUEUE_SEMAPHORE	Occurs when a thread from the dispatcher pool is waiting for more work to process. The wait time for this wait type is expected to increase when the dispatcher is idle.
DLL_LOADING_MUTEX	Occurs once while waiting for the XML parser DLL to load.
DROPTEMP	Occurs between attempts to drop a temporary object if the previous attempt failed. The wait duration grows exponentially with each failed drop attempt.
DTC	Occurs when a task is waiting on an event that is used to manage state transition. This state controls when the recovery of Microsoft Distributed Transaction Coordinator (MS DTC) transactions occurs after SQL Server receives notification that the MS DTC service has become unavailable. This state also describes a task that is waiting when a commit of a MS DTC transaction is initiated by SQL Server and SQL Server is waiting for the MS DTC commit to finish.
DTC_ABORT_REQUEST	Occurs in a MS DTC worker session when the session is waiting to take ownership of a MS DTC transaction. After MS DTC owns the transaction, the session can roll back the transaction. Generally, the session will wait for another session that is using the transaction.
DTC_RESOLVE	Occurs when a recovery task is waiting for the master database in a cross-database transaction so that the task can query the outcome of the transaction.

WAIT TYPE	DESCRIPTION
DTC_STATE	Occurs when a task is waiting on an event that protects changes to the internal MS DTC global state object. This state should be held for very short periods of time.
DTC_TMDOWN_REQUEST	Occurs in a MS DTC worker session when SQL Server receives notification that the MS DTC service is not available. First, the worker will wait for the MS DTC recovery process to start. Then, the worker waits to obtain the outcome of the distributed transaction that the worker is working on. This may continue until the connection with the MS DTC service has been reestablished.
DTC_WAITFOR_OUTCOME	Occurs when recovery tasks wait for MS DTC to become active to enable the resolution of prepared transactions.
DUMP_LOG_COORDINATOR	Occurs when a main task is waiting for a subtask to generate data. Ordinarily, this state does not occur. A long wait indicates an unexpected blockage. The subtask should be investigated.
DUMPTRIGGER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
EC	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
EE_PMOLOCK	Occurs during synchronization of certain types of memory allocations during statement execution.
EE_SPECPROC_MAP_INIT	Occurs during synchronization of internal procedure hash table creation. This wait can only occur during the initial accessing of the hash table after the SQL Server instance starts.
ENABLE_VERSIONING	Occurs when SQL Server waits for all update transactions in this database to finish before declaring the database ready to transition to snapshot isolation allowed state. This state is used when SQL Server enables snapshot isolation by using the ALTER DATABASE statement.
ERROR_REPORTING_MANAGER	Occurs during synchronization of multiple concurrent error log initializations.
EXCHANGE	Occurs during synchronization in the query processor exchange iterator during parallel queries.
EXECSYNC	Occurs during parallel queries while synchronizing in query processor in areas not related to the exchange iterator. Examples of such areas are bitmaps, large binary objects (LOBs), and the spool iterator. LOBs may frequently use this wait state.
EXECUTION_PIPE_EVENT_INTERNAL	Occurs during synchronization between producer and consumer parts of batch execution that are submitted through the connection context.

WAIT TYPE	DESCRIPTION
FAILPOINT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
FCB_REPLICA_READ	Occurs when the reads of a snapshot (or a temporary snapshot created by DBCC) sparse file are synchronized.
FCB_REPLICA_WRITE	Occurs when the pushing or pulling of a page to a snapshot (or a temporary snapshot created by DBCC) sparse file is synchronized.
FS_FC_RWLOCK	Occurs when there is a wait by the FILESTREAM garbage collector to do either of the following: Disable garbage collection (used by backup and restore). Execute one cycle of the FILESTREAM garbage collector.
FS_GARBAGE_COLLECTOR_SHUTDOWN	Occurs when the FILESTREAM garbage collector is waiting for cleanup tasks to be completed.
FS_HEADER_RWLOCK	Occurs when there is a wait to acquire access to the FILESTREAM header of a FILESTREAM data container to either read or update contents in the FILESTREAM header file (Filestream.hdr).
FS_LOGTRUNC_RWLOCK	Occurs when there is a wait to acquire access to FILESTREAM log truncation to do either of the following: Temporarily disable FILESTREAM log (FSLOG) truncation (used by backup and restore). Execute one cycle of FSLOG truncation.
FSA_FORCE_OWN_XACT	Occurs when a FILESTREAM file I/O operation needs to bind to the associated transaction, but the transaction is currently owned by another session.
FSAGENT	Occurs when a FILESTREAM file I/O operation is waiting for a FILESTREAM agent resource that is being used by another file I/O operation.
FSTR_CONFIG_MUTEX	Occurs when there is a wait for another FILESTREAM feature reconfiguration to be completed.
FSTR_CONFIG_RWLOCK	Occurs when there is a wait to serialize access to the FILESTREAM configuration parameters.
FT_METADATA_MUTEX	Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_RESTART_CRAWL	Occurs when a full-text crawl needs to restart from a last known good point to recover from a transient failure. The wait lets the worker tasks currently working on that population to complete or exit the current step.

WAIT TYPE	DESCRIPTION
FULLTEXT GATHERER	Occurs during synchronization of full-text operations.
GUARDIAN	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
HTTP_ENUMERATION	Occurs at startup to enumerate the HTTP endpoints to start HTTP.
HTTP_START	Occurs when a connection is waiting for HTTP to complete initialization.
IMPPROV_IOWAIT	Occurs when SQL Server waits for a bulkload I/O to finish.
INTERNAL_TESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
IO_AUDIT_MUTEX	Occurs during synchronization of trace event buffers.
IO_COMPLETION	Occurs while waiting for I/O operations to complete. This wait type generally represents non-data page I/Os. Data page I/O completion waits appear as PAGEIOLATCH_* waits.
IO_QUEUE_LIMIT	Occurs when the asynchronous IO queue for the Azure SQL Database has too many IOs pending. Tasks trying to issue another IO are blocked on this wait type until the number of pending IOs drop below the threshold. The threshold is proportional to the DTUs assigned to the database.
IO_RETRY	Occurs when an I/O operation such as a read or a write to disk fails because of insufficient resources, and is then retried.
IOAFF_RANGE_QUEUE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
KSOURCE_WAKEUP	Used by the service control task while waiting for requests from the Service Control Manager. Long waits are expected and do not indicate a problem.
KTM_ENLISTMENT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
KTM_RECOVERY_MANAGER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
KTM_RECOVERY_RESOLUTION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
LATCH_DT	Occurs when waiting for a DT (destroy) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.

WAIT TYPE	DESCRIPTION
LATCH_EX	Occurs when waiting for an EX (exclusive) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.
LATCH_KP	Occurs when waiting for a KP (keep) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.
LATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
LATCH_SH	Occurs when waiting for an SH (share) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.
LATCH_UP	Occurs when waiting for an UP (update) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.
LAZYWRITER_SLEEP	Occurs when lazywriter tasks are suspended. This is a measure of the time spent by background tasks that are waiting. Do not consider this state when you are looking for user stalls.
LCK_M_BU	Occurs when a task is waiting to acquire a Bulk Update (BU) lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_IS	Occurs when a task is waiting to acquire an Intent Shared (IS) lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_IU	Occurs when a task is waiting to acquire an Intent Update (IU) lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_IX	Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_RIn_NL	Occurs when a task is waiting to acquire a NULL lock on the current key value, and an Insert Range lock between the current and previous key. A NULL lock on the key is an instant release lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).

WAIT TYPE	DESCRIPTION
LCK_M_RIn_S	Occurs when a task is waiting to acquire a shared lock on the current key value, and an Insert Range lock between the current and previous key. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_RIn_U	Task is waiting to acquire an Update lock on the current key value, and an Insert Range lock between the current and previous key. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_RIn_X	Occurs when a task is waiting to acquire an Exclusive lock on the current key value, and an Insert Range lock between the current and previous key. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_RS_S	Occurs when a task is waiting to acquire a Shared lock on the current key value, and a Shared Range lock between the current and previous key. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_RS_U	Occurs when a task is waiting to acquire an Update lock on the current key value, and an Update Range lock between the current and previous key. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_RX_S	Occurs when a task is waiting to acquire a Shared lock on the current key value, and an Exclusive Range lock between the current and previous key. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_RX_U	Occurs when a task is waiting to acquire an Update lock on the current key value, and an Exclusive range lock between the current and previous key. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_RX_X	Occurs when a task is waiting to acquire an Exclusive lock on the current key value, and an Exclusive Range lock between the current and previous key. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_S	Occurs when a task is waiting to acquire a Shared lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_SCH_M	Occurs when a task is waiting to acquire a Schema Modify lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_SCH_S	Occurs when a task is waiting to acquire a Schema Share lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_SIU	Occurs when a task is waiting to acquire a Shared With Intent Update lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).

WAIT TYPE	DESCRIPTION
LCK_M_SIX	Occurs when a task is waiting to acquire a Shared With Intent Exclusive lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_U	Occurs when a task is waiting to acquire an Update lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_UIX	Occurs when a task is waiting to acquire an Update With Intent Exclusive lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LCK_M_X	Occurs when a task is waiting to acquire an Exclusive lock. For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).
LOG_RATE_GOVERNOR	Occurs when DB is waiting for quota to write to the log.
LOGBUFFER	Occurs when a task is waiting for space in the log buffer to store a log record. Consistently high values may indicate that the log devices cannot keep up with the amount of log being generated by the server.
LOGGENERATION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
LOGMGR	Occurs when a task is waiting for any outstanding log I/Os to finish before shutting down the log while closing the database.
LOGMGR_FLUSH	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
LOGMGR_QUEUE	Occurs while the log writer task waits for work requests.
LOGMGR_RESERVE_APPEND	Occurs when a task is waiting to see whether log truncation frees up log space to enable the task to write a new log record. Consider increasing the size of the log file(s) for the affected database to reduce this wait.
LOWFAIL_MEMMGR_QUEUE	Occurs while waiting for memory to be available for use.
MSQL_DQ	Occurs when a task is waiting for a distributed query operation to finish. This is used to detect potential Multiple Active Result Set (MARS) application deadlocks. The wait ends when the distributed query call finishes.
MSQL_XACT_MGR_MUTEX	Occurs when a task is waiting to obtain ownership of the session transaction manager to perform a session level transaction operation.
MSQL_XACT_MUTEX	Occurs during synchronization of transaction usage. A request must acquire the mutex before it can use the transaction.

WAIT TYPE	DESCRIPTION
MSQL_XP	Occurs when a task is waiting for an extended stored procedure to end. SQL Server uses this wait state to detect potential MARS application deadlocks. The wait stops when the extended stored procedure call ends.
MSSEARCH	Occurs during Full-Text Search calls. This wait ends when the full-text operation completes. It does not indicate contention, but rather the duration of full-text operations.
NET_WAITFOR_PACKET	Occurs when a connection is waiting for a network packet during a network read.
OLEDB	Occurs when SQL Server calls the SQL Server Native Client OLE DB Provider. This wait type is not used for synchronization. Instead, it indicates the duration of calls to the OLE DB provider.
ONDEMAND_TASK_QUEUE	Occurs while a background task waits for high priority system task requests. Long wait times indicate that there have been no high priority requests to process, and should not cause concern.
PAGEIOLATCH_DT	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Destroy mode. Long waits may indicate problems with the disk subsystem.
PAGEIOLATCH_EX	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Exclusive mode. Long waits may indicate problems with the disk subsystem.
PAGEIOLATCH_KP	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Keep mode. Long waits may indicate problems with the disk subsystem.
PAGEIOLATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PAGEIOLATCH_SH	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Shared mode. Long waits may indicate problems with the disk subsystem.
PAGEIOLATCH_UP	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Update mode. Long waits may indicate problems with the disk subsystem.
PAGELATCH_DT	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Destroy mode.
PAGELATCH_EX	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Exclusive mode.
PAGELATCH_KP	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Keep mode.

WAIT TYPE	DESCRIPTION
PAGELATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PAGELATCH_SH	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Shared mode.
PAGELATCH_UP	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Update mode.
PARALLEL_BACKUP_QUEUE	Occurs when serializing output produced by RESTORE HEADERONLY, RESTORE FILELISTONLY, or RESTORE LABELONLY.
PREEMPTIVE_ABR	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PREEMPTIVE_AUDIT_ACCESS_EVENTLOG	Occurs when the SQL Server Operating System (SQLOS) scheduler switches to preemptive mode to write an audit event to the Windows event log.
PREEMPTIVE_AUDIT_ACCESS_SECLOG	Occurs when the SQLOS scheduler switches to preemptive mode to write an audit event to the Windows Security log.
PREEMPTIVE_CLOSEBACKUPMEDIA	Occurs when the SQLOS scheduler switches to preemptive mode to close backup media.
PREEMPTIVE_CLOSEBACKUPTAPE	Occurs when the SQLOS scheduler switches to preemptive mode to close a tape backup device.
PREEMPTIVE_CLOSEBACKUPVDIDEVICE	Occurs when the SQLOS scheduler switches to preemptive mode to close a virtual backup device.
PREEMPTIVE_CLUSAPI_CLUSTERRESOURCECONTROL	Occurs when the SQLOS scheduler switches to preemptive mode to perform Windows failover cluster operations.
PREEMPTIVE_COM_COCREATEINSTANCE	Occurs when the SQLOS scheduler switches to preemptive mode to create a COM object.
PREEMPTIVE_HADR_LEASE_MECHANISM	Always On Availability Groups lease manager scheduling for CSS diagnostics.
PREEMPTIVE_SOSTESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PREEMPTIVE_STRESSDRIVER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PREEMPTIVE_TESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PREEMPTIVE_XETESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

WAIT TYPE	DESCRIPTION
PRINT_ROLLBACK_PROGRESS	Used to wait while user processes are ended in a database that has been transitioned by using the ALTER DATABASE termination clause. For more information, see ALTER DATABASE (Transact-SQL).
PWAIT_HADR_CHANGE_NOTIFIER_TERMINATION_SYNC	Occurs when a background task is waiting for the termination of the background task that receives (via polling) Windows Server Failover Clustering notifications. Internal use only.
PWAIT_HADR_CLUSTER_INTEGRATION	An append, replace, and/or remove operation is waiting to grab a write lock on an Always On internal list (such as a list of networks, network addresses, or availability group listeners). Internal use only.
PWAIT_HADR_OFFLINE_COMPLETED	An Always On drop availability group operation is waiting for the target availability group to go offline before destroying Windows Server Failover Clustering objects.
PWAIT_HADR_ONLINE_COMPLETED	An Always On create or failover availability group operation is waiting for the target availability group to come online.
PWAIT_HADR_POST_ONLINE_COMPLETED	An Always On drop availability group operation is waiting for the termination of any background task that was scheduled as part of a previous command. For example, there may be a background task that is transitioning availability databases to the primary role. The DROP AVAILABILITY GROUP DDL must wait for this background task to terminate in order to avoid race conditions.
PWAIT_HADR_WORKITEM_COMPLETED	Internal wait by a thread waiting for an async work task to complete. This is an expected wait and is for CSS use.
PWAIT_MD_LOGIN_STATS	Occurs during internal synchronization in metadata on login stats.
PWAIT_MD_RELATION_CACHE	Occurs during internal synchronization in metadata on table or index.
PWAIT_MD_SERVER_CACHE	Occurs during internal synchronization in metadata on linked servers.
PWAIT_MD_UPGRADE_CONFIG	Occurs during internal synchronization in upgrading server wide configurations.
PWAIT_METADATA_LAZYCACHE_RWLOCk	Occurs during internal synchronization in metadata cache along with iterating index or stats in a table.
QPJOB_KILL	Indicates that an asynchronous automatic statistics update was canceled by a call to KILL as the update was starting to run. The terminating thread is suspended, waiting for it to start listening for KILL commands. A good value is less than one second.

WAIT TYPE	DESCRIPTION
QPJOB_WAITFOR_ABORT	Indicates that an asynchronous automatic statistics update was canceled by a call to KILL when it was running. The update has now completed but is suspended until the terminating thread message coordination is complete. This is an ordinary but rare state, and should be very short. A good value is less than one second.
QRY_MEM_GRANT_INFO_MUTEX	Occurs when Query Execution memory management tries to control access to static grant information list. This state lists information about the current granted and waiting memory requests. This state is a simple access control state. There should never be a long wait on this state. If this mutex is not released, all new memory-using queries will stop responding.
QUERY_ERRHDL_SERVICE_DONE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
QUERY_EXECUTION_INDEX_SORT_EVENT_OPEN	Occurs in certain cases when offline create index build is run in parallel, and the different worker threads that are sorting synchronize access to the sort files.
QUERY_NOTIFICATION_MGR_MUTEX	Occurs during synchronization of the garbage collection queue in the Query Notification Manager.
QUERY_NOTIFICATION_SUBSCRIPTION_MUTEX	Occurs during state synchronization for transactions in Query Notifications.
QUERY_NOTIFICATION_TABLE_MGR_MUTEX	Occurs during internal synchronization within the Query Notification Manager.
QUERY_NOTIFICATION_UNITTEST_MUTEX	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
QUERY_OPTIMIZER_PRINT_MUTEX	Occurs during synchronization of query optimizer diagnostic output production. This wait type only occurs if diagnostic settings have been enabled under direction of Microsoft Product Support.
QUERY_TRACEOUT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
QUERY_WAIT_ERRHDL_SERVICE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
RECOVER_CHANGEDB	Occurs during synchronization of database status in warm standby database.
REPL_CACHE_ACCESS	Occurs during synchronization on a replication article cache. During these waits, the replication log reader stalls, and data definition language (DDL) statements on a published table are blocked.

WAIT TYPE	DESCRIPTION
REPL_SCHEMA_ACCESS	Occurs during synchronization of replication schema version information. This state exists when DDL statements are executed on the replicated object, and when the log reader builds or consumes versioned schema based on DDL occurrence.
REPLICA_WRITES	Occurs while a task waits for completion of page writes to database snapshots or DBCC replicas.
REQUEST_DISPENSER_PAUSE	Occurs when a task is waiting for all outstanding I/O to complete, so that I/O to a file can be frozen for snapshot backup.
REQUEST_FOR_DEADLOCK_SEARCH	Occurs while the deadlock monitor waits to start the next deadlock search. This wait is expected between deadlock detections, and lengthy total waiting time on this resource does not indicate a problem.
RESMGR_THROTTLED	Occurs when a new request comes in and is throttled based on the GROUP_MAX_REQUESTS setting.
RESOURCE_QUEUE	Occurs during synchronization of various internal resource queues.
RESOURCE_SEMAPHORE	Occurs when a query memory request cannot be granted immediately due to other concurrent queries. High waits and wait times may indicate excessive number of concurrent queries, or excessive memory request amounts.
RESOURCE_SEMAPHORE_MUTEX	Occurs while a query waits for its request for a thread reservation to be fulfilled. It also occurs when synchronizing query compile and memory grant requests.
RESOURCE_SEMAPHORE_QUERY_COMPILE	Occurs when the number of concurrent query compilations reaches a throttling limit. High waits and wait times may indicate excessive compilations, recompiles, or uncachable plans.
RESOURCE_SEMAPHORE_SMALL_QUERY	Occurs when memory request by a small query cannot be granted immediately due to other concurrent queries. Wait time should not exceed more than a few seconds, because the server transfers the request to the main query memory pool if it fails to grant the requested memory within a few seconds. High waits may indicate an excessive number of concurrent small queries while the main memory pool is blocked by waiting queries.
SE_REPL_CATCHUP_THROTTLE	Occurs when the transaction is waiting for one of the database secondaries to make progress.
SE_REPL_COMMIT_ACK	Occurs when the transaction is waiting for quorum commit acknowledgement from secondary replicas.
SE_REPL_COMMIT_TURN	Occurs when the transaction is waiting for commit after receiving quorum commit acknowledgements.

WAIT TYPE	DESCRIPTION
SE_REPL_ROLLBACK_ACK	Occurs when the transaction is waiting for quorum rollback acknowledgement from secondary replicas.
SE_REPL_SLOW_SECONDARY_THROTTLE	Occurs when the thread is waiting for one of the database secondary replicas.
SEC_DROP_TEMP_KEY	Occurs after a failed attempt to drop a temporary security key before a retry attempt.
SECURITY_MUTEX	Occurs when there is a wait for mutexes that control access to the global list of Extensible Key Management (EKM) cryptographic providers and the session-scoped list of EKM sessions.
SEQUENTIAL_GUID	Occurs while a new sequential GUID is being obtained.
SERVER_IDLE_CHECK	Occurs during synchronization of SQL Server instance idle status when a resource monitor is attempting to declare a SQL Server instance as idle or trying to wake up.
SHUTDOWN	Occurs while a shutdown statement waits for active connections to exit.
SLEEP_BPOOL_FLUSH	Occurs when a checkpoint is throttling the issuance of new I/Os in order to avoid flooding the disk subsystem.
SLEEP_DBSTARTUP	Occurs during database startup while waiting for all databases to recover.
SLEEP_DCOMSTARTUP	Occurs once at most during SQL Server instance startup while waiting for DCOM initialization to complete.
SLEEP_MSDBSTARTUP	Occurs when SQL Trace waits for the msdb database to complete startup.
SLEEP_SYSTEMTASK	Occurs during the start of a background task while waiting for tempdb to complete startup.
SLEEP_TASK	Occurs when a task sleeps while waiting for a generic event to occur.
SLEEP_TEMPDBSTARTUP	Occurs while a task waits for tempdb to complete startup.
SNI_CRITICAL_SECTION	Occurs during internal synchronization within SQL Server networking components.
SNI_HTTP_WAITFOR_0_DISCON	Occurs during SQL Server shutdown, while waiting for outstanding HTTP connections to exit.
SNI_LISTENER_ACCESS	Occurs while waiting for non-uniform memory access (NUMA) nodes to update state change. Access to state change is serialized.

WAIT TYPE	DESCRIPTION
SNI_TASK_COMPLETION	Occurs when there is a wait for all tasks to finish during a NUMA node state change.
SOAP_READ	Occurs while waiting for an HTTP network read to complete.
SOAP_WRITE	Occurs while waiting for an HTTP network write to complete.
SOS_CALLBACK_REMOVAL	Occurs while performing synchronization on a callback list in order to remove a callback. It is not expected for this counter to change after server initialization is completed.
SOS_DISPATCHER_MUTEX	Occurs during internal synchronization of the dispatcher pool. This includes when the pool is being adjusted.
SOS_LOCALALLOCATORLIST	Occurs during internal synchronization in the SQL Server memory manager.
SOS_MEMORY_USAGE_ADJUSTMENT	Occurs when memory usage is being adjusted among pools.
SOS_OBJECT_STORE_DESTROY_MUTEX	Occurs during internal synchronization in memory pools when destroying objects from the pool.
SOS_PROCESS_AFFINITY_MUTEX	Occurs during synchronizing of access to process affinity settings.
SOS_RESERVEDMEMBLOCKLIST	Occurs during internal synchronization in the SQL Server memory manager.
SOS_SCHEDULER_YIELD	Occurs when a task voluntarily yields the scheduler for other tasks to execute. During this wait the task is waiting for its quantum to be renewed.
SOS_SMALL_PAGE_ALLOC	Occurs during the allocation and freeing of memory that is managed by some memory objects.
SOS_STACKSTORE_INIT_MUTEX	Occurs during synchronization of internal store initialization.
SOS_SYNC_TASK_ENQUEUE_EVENT	Occurs when a task is started in a synchronous manner. Most tasks in SQL Server are started in an asynchronous manner, in which control returns to the starter immediately after the task request has been placed on the work queue.
SOS_VIRTUALMEMORY_LOW	Occurs when a memory allocation waits for a resource manager to free up virtual memory.
SOSHOST_EVENT	Occurs when a hosted component, such as CLR, waits on a SQL Server event synchronization object.
SOSHOST_INTERNAL	Occurs during synchronization of memory manager callbacks used by hosted components, such as CLR.
SOSHOST_MUTEX	Occurs when a hosted component, such as CLR, waits on a SQL Server mutex synchronization object.

WAIT TYPE	DESCRIPTION
SOSHOST_RWLOCK	Occurs when a hosted component, such as CLR, waits on a SQL Server reader-writer synchronization object.
SOSHOST_SEMAPHORE	Occurs when a hosted component, such as CLR, waits on a SQL Server semaphore synchronization object.
SOSHOST_SLEEP	Occurs when a hosted task sleeps while waiting for a generic event to occur. Hosted tasks are used by hosted components such as CLR.
SOSHOST_TRACELOCK	Occurs during synchronization of access to trace streams.
SOSHOST_WAITFORDONE	Occurs when a hosted component, such as CLR, waits for a task to complete.
SQLCLR_APPDOMAIN	Occurs while CLR waits for an application domain to complete startup.
SQLCLR_ASSEMBLY	Occurs while waiting for access to the loaded assembly list in the appdomain.
SQLCLR_DEADLOCK_DETECTION	Occurs while CLR waits for deadlock detection to complete.
SQLCLR_QUANTUM_PUNISHMENT	Occurs when a CLR task is throttled because it has exceeded its execution quantum. This throttling is done in order to reduce the effect of this resource-intensive task on other tasks.
SQLSORT_NORMMUTEX	Occurs during internal synchronization, while initializing internal sorting structures.
SQLSORT_SORTMUTEX	Occurs during internal synchronization, while initializing internal sorting structures.
SQLTRACE_BUFFER_FLUSH	Occurs when a task is waiting for a background task to flush trace buffers to disk every four seconds.
SQLTRACE_LOCK	Occurs during synchronization on trace buffers during a file trace.
SQLTRACE_SHUTDOWN	Occurs while trace shutdown waits for outstanding trace events to complete.
SQLTRACE_WAIT_ENTRIES	Occurs while a SQL Trace event queue waits for packets to arrive on the queue.
SRVPROC_SHUTDOWN	Occurs while the shutdown process waits for internal resources to be released to shutdown cleanly.
TEMPOBJ	Occurs when temporary object drops are synchronized. This wait is rare, and only occurs if a task has requested exclusive access for temp table drops.

WAIT TYPE	DESCRIPTION
THREADPOOL	Occurs when a task is waiting for a worker to run on. This can indicate that the maximum worker setting is too low, or that batch executions are taking unusually long, thus reducing the number of workers available to satisfy other batches.
TIMEPRIV_TIMEPERIOD	Occurs during internal synchronization of the Extended Events timer.
TRACEWRITE	Occurs when the SQL Trace rowset trace provider waits for either a free buffer or a buffer with events to process.
TRAN_MARKLATCH_DT	Occurs when waiting for a destroy mode latch on a transaction mark latch. Transaction mark latches are used for synchronization of commits with marked transactions.
TRAN_MARKLATCH_EX	Occurs when waiting for an exclusive mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions.
TRAN_MARKLATCH_KP	Occurs when waiting for a keep mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions.
TRAN_MARKLATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
TRAN_MARKLATCH_SH	Occurs when waiting for a shared mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions.
TRAN_MARKLATCH_UP	Occurs when waiting for an update mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions.
TRANSACTION_MUTEX	Occurs during synchronization of access to a transaction by multiple batches.
UTIL_PAGE_ALLOC	Occurs when transaction log scans wait for memory to be available during memory pressure.
VIA_ACCEPT	Occurs when a Virtual Interface Adapter (VIA) provider connection is completed during startup.
VIEW_DEFINITION_MUTEX	Occurs during synchronization on access to cached view definitions.
WAIT_FOR_RESULTS	Occurs when waiting for a query notification to be triggered.
WAITFOR	Occurs as a result of a WAITFOR Transact-SQL statement. The duration of the wait is determined by the parameters to the statement. This is a user-initiated wait.
WAITFOR_TASKSHUTDOWN	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

WAIT TYPE	DESCRIPTION
WAITSTAT_MUTEX	Occurs during synchronization of access to the collection of statistics used to populate sys.dm_os_wait_stats.
WCC	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
WORKTBL_DROP	Occurs while pausing before retrying, after a failed worktable drop.
WRITE_COMPLETION	Occurs when a write operation is in progress.
WRITELOG	Occurs while waiting for a log flush to complete. Common operations that cause log flushes are checkpoints and transaction commits.
XACT_OWN_TRANSACTION	Occurs while waiting to acquire ownership of a transaction.
XACT_RECLAIM_SESSION	Occurs while waiting for the current owner of a session to release ownership of the session.
XACTLOCKINFO	Occurs during synchronization of access to the list of locks for a transaction. In addition to the transaction itself, the list of locks is accessed by operations such as deadlock detection and lock migration during page splits.
XACTWORKSPACE_MUTEX	Occurs during synchronization of defections from a transaction, as well as the number of database locks between enlist members of a transaction.
XE_BUFFERMGR_ALLPROCESSED_EVENT	Occurs when Extended Events session buffers are flushed to targets. This wait occurs on a background thread.
XE_BUFFERMGR_FREEBUF_EVENT	Occurs when either of the following conditions is true: An Extended Events session is configured for no event loss, and all buffers in the session are currently full. This can indicate that the buffers for an Extended Events session are too small, or should be partitioned. Audits experience a delay. This can indicate a disk bottleneck on the drive where the audits are written.
XE_DISPATCHER_CONFIG_SESSION_LIST	Occurs when an Extended Events session that is using asynchronous targets is started or stopped. This wait indicates either of the following: An Extended Events session is registering with a background thread pool. The background thread pool is calculating the required number of threads based on current load.
XE_DISPATCHER_JOIN	Occurs when a background thread that is used for Extended Events sessions is terminating.

WAIT TYPE	DESCRIPTION
XE_DISPATCHER_WAIT	Occurs when a background thread that is used for Extended Events sessions is waiting for event buffers to process.
XE_MODULEMGR_SYNC	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
XE_OLS_LOCK	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
XE_PACKAGE_LOCK_BACKOFF	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_COMPROWSET_RWLOCK	Full-text is waiting on fragment metadata operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_IFTS_RWLOCK	Full-text is waiting on internal synchronization. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_IFTS_SCHEDULER_IDLE_WAIT	Full-text scheduler sleep wait type. The scheduler is idle.
FT_IFTSHC_MUTEX	Full-text is waiting on an fdhost control operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_IFTSISM_MUTEX	Full-text is waiting on communication operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_MASTER_MERGE	Full-text is waiting on master merge operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.

sys.dm_operation_status (Azure SQL Database)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about operations performed on databases in a Azure SQL Database server.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_activity_id	**uniqueidentifier**	ID of the operation. Not null.
resource_type	**int**	Denotes the type of resource on which the operation is performed. Not null. In the current release, this view tracks operations performed on SQL Database only, and the corresponding integer value is 0.
resource_type_desc	**nvarchar(2048)**	Description of the resource type on which the operation is performed. In the current release, this view tracks operations performed on SQL Database only.
major_resource_id	**sql_variant**	Name of the SQL Database on which the operation is performed. Not Null.
minor_resource_id	**sql_variant**	For internal use only. Not null.
operation	**nvarchar(60)**	Operation performed on a SQL Database, such as CREATE or ALTER.
state	**tinyint**	The state of the operation. 0 = Pending 1 = In progress 2 = Completed 3 = Failed 4 = Cancelled
state_desc	**nvarchar(120)**	PENDING = operation is waiting for resource or quota availability. IN_PROGRESS = operation has started and is in progress. COMPLETED = operation completed successfully. FAILED = operation failed. See the **error_desc** column for details. CANCELLED = operation stopped at the request of the user.

COLUMN NAME	DATA TYPE	DESCRIPTION
percent_complete	**int**	Percentage of operation that has completed. Values are not continuous and the valid values are listed below. Not NULL. 0 = Operation not started 50 = Operation in progress 100 = Operation complete
error_code	**int**	Code indicating the error that occurred during a failed operation. If the value is 0, it indicates that the operation completed successfully.
error_desc	**nvarchar(2048)**	Description of the error that occurred during a failed operation.
error_severity	**int**	Severity level of the error that occurred during a failed operation. For more information about error severities, see Database Engine Error Severities.
error_state	**int**	Reserved for future use. Future compatibility is not guaranteed.
start_time	**datetime**	Timestamp when the operation started.
last_modify_time	**datetime**	Timestamp when the record was last modified for a long running operation. In case of successfully completed operations, this field displays the timestamp when the operation completed.

Permissions

This view is only available in the **master** database to the server-level principal login.

Remarks

To use this view, you must be connected to the **master** database. Use the `sys.dm_operation_status` view in the **master** database of the SQL Database server to track the status of the following operations performed on a SQL Database:

- Create database
- Copy database. Database Copy creates a record in this view on both the source and target servers.
- Alter database
- Change the performance level of a service tier
- Change the service tier of a database, such as changing from Basic to Standard.
- Setting up a Geo-Replication relationship
- Terminating a Geo-Replication relationship

- Restore database
- Delete database

Example

Show most recent geo-replication operations associated with database ‘mydb’.

```
SELECT * FROM sys.dm_ operation_status
   WHERE major_resource_id = ‘myddb’
   ORDER BY start_time DESC;
```

See Also

Geo-Replication Dynamic Management Views and Functions (Azure SQL Database)
sys.dm_geo_replication_link_status (Azure SQL Database)
sys.geo_replication_links (Azure SQL Database)
ALTER DATABASE (Azure SQL Database)

sys.dm_database_copies (Azure SQL Database)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the database copy.

To return information about geo-replication links, use the sys.geo_replication_links or sys.dm_geo_replication_link_status views (available in SQL Database V12).

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	The ID of the current database in the `sys.databases` view.
start_date	**datetimeoffset**	The UTC time at a regional SQL Database datacenter when the database copying was initiated.
modify_date	**datetimeoffset**	The UTC time at regional SQL Database datacenter when the database copying has completed. The new database is transactionally consistent with the primary database as of this time. The completion information is updated every 1 minute. UTC time reflecting the last update of the percent_complete field.
percent_complete	**real**	The percentage of bytes that have been copied. Values range from 0 to 100. SQL Database may automatically recover from some errors, such as failover, and restart the database copy. In this case, percent_complete would restart from 0.
error_code	**int**	When greater than 0, the code indicating the error that has occurred while copying. Value equals 0 if no errors have occurred.
error_desc	**nvarchar(4096)**	Description of the error that occurred while copying.
error_severity	**int**	Returns 16 if the database copy failed.
error_state	**int**	Returns 1 if copy failed.
copy_guid	**uniqueidentifier**	Unique ID of the copy operation.

COLUMN NAME	DATA TYPE	DESCRIPTION
partner_server	**sysname**	Name of the SQL Database server where the copy is created.
partner_database	**sysname**	Name of the database copy on the partner server.
replication_state	**tinyint**	The state of continuous-copy replication for this database. Values are: 0=Pending. Creation of the database copy is scheduled but the necessary preparation steps are not yet completed or are temporarily blocked by the seeding quota. 1=Seeding. The copy database being seeded is not yet fully synchronized with the source database. In this state you cannot connect to the copy. To cancel the seeding operation in progress, the copy database must be dropped.
replication_state_desc	**nvarchar(256)**	Description of replication_state, one of: PENDING SEEDING
maximum_lag	**int**	Reserved field.
is_continuous_copy	**bit**	0 = Returns 0
is_target_role	**bit**	0 =Source database 1 = Copy database
is_interlink_connected	bit	Reserved field.
is_offline_secondary	bit	Reserved field.

Permissions

This view is only available in the **master** database to the server-level principal login.

Remarks

You can use the **sys.dm_database_copies** view in the **master** database of the source or target SQL Database server. When the database copy completes successfully and the new database becomes ONLINE, the row in the **sys.dm_database_copies** view is removed automatically.

sys.dm_db_resource_stats (Azure SQL Database)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns CPU, I/O, and memory consumption for an Azure SQL Database database. One row exists for every 15 seconds, even if there is no activity in the database. Historical data is maintained for one hour.

COLUMNS	DATA TYPE	DESCRIPTION
end_time	**datetime**	UTC time indicates the end of the current reporting interval.
avg_cpu_percent	**decimal (5,2)**	Average compute utilization in percentage of the limit of the service tier.
avg_data_io_percent	**decimal (5,2)**	Average data I/O utilization in percentage of the limit of the service tier.
avg_log_write_percent	**decimal (5,2)**	Average write resource utilization in percentage of the limit of the service tier.
avg_memory_usage_percent	**decimal (5,2)**	Average memory utilization in percentage of the limit of the service tier. This includes memory used for storage of In-Memory OLTP objects.
xtp_storage_percent	**decimal (5,2)**	Storage utilization for In-Memory OLTP in percentage of the limit of the service tier (at the end of the reporting interval). This includes memory used for storage of the following In-Memory OLTP objects: memory-optimized tables, indexes, and table variables. It also includes memory used for processing ALTER TABLE operations. Returns 0 if In-Memory OLTP is not used in the database.
max_worker_percent	**decimal (5,2)**	Maximum concurrent workers (requests) in percentage of the limit of the database's service tier.
max_session_percent	**decimal (5,2)**	Maximum concurrent sessions in percentage of the limit of the database's service tier.

COLUMNS	DATA TYPE	DESCRIPTION
dtu_limit	**int**	Current max database DTU setting for this database during this interval.

> **TIP**
> For more context about these limits and service tiers, see the topics Service Tiers and Service tier capabilities and limits.

Permissions

This view requires VIEW DATABASE STATE permission.

Remarks

The data returned by **sys.dm_db_resource_stats** is expressed as a percentage of the maximum allowed limits for the service tier/performance level that you are running.

If the database was failed over to another server within the last 60 minutes, the view will only return data for the time it has been the primary database since that failover.

For a less granular view of this data, use **sys.resource_stats** catalog view in the **master** database. This view captures data every 5 minutes and maintains historical data for 14 days. For more information, see sys.resource_stats (Azure SQL Database).

When a database is a member of an elastic pool, resource statistics presented as percent values, are expressed as the percent of the max limit for the databases as set in the elastic pool configuration.

Example

The following example returns resource utilization data ordered by the most recent time for the currently connected database.

```
SELECT * FROM sys.dm_db_resource_stats ORDER BY end_time DESC;
```

The following example identifies the average DTU consumption in terms of a percentage of the maximum allowed DTU limit in the performance level for the user database over the past hour. Consider increasing the performance level as these percentages near 100% on a consistent basis.

```
SELECT end_time,
  (SELECT Max(v)
   FROM (VALUES (avg_cpu_percent), (avg_data_io_percent), (avg_log_write_percent)) AS
   value(v)) AS [avg_DTU_percent]
FROM sys.dm_db_resource_stats;
```

The following example returns the average and maximum values for CPU percent, data and log I/O, and memory consumption over the last hour.

```
SELECT
    AVG(avg_cpu_percent) AS 'Average CPU Utilization In Percent',
    MAX(avg_cpu_percent) AS 'Maximum CPU Utilization In Percent',
    AVG(avg_data_io_percent) AS 'Average Data IO In Percent',
    MAX(avg_data_io_percent) AS 'Maximum Data IO In Percent',
    AVG(avg_log_write_percent) AS 'Average Log Write Utilization In Percent',
    MAX(avg_log_write_percent) AS 'Maximum Log Write Utilization In Percent',
    AVG(avg_memory_usage_percent) AS 'Average Memory Usage In Percent',
    MAX(avg_memory_usage_percent) AS 'Maximum Memory Usage In Percent'
FROM sys.dm_db_resource_stats;
```

See Also

sys.resource_stats (Azure SQL Database)
Service Tiers
Service tier capabilities and limits

Database Mirroring - sys.dm_db_mirroring_auto_page_repair

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for every automatic page-repair attempt on any mirrored database on the server instance. This view contains rows for the latest automatic page-repair attempts on a given mirrored database, with a maximum of 100 rows per database. As soon as a database reaches the maximum, the row for its next automatic page-repair attempt replaces one of the existing entries. The following table defines the meaning of the various columns.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	ID of the database to which this row corresponds.
file_id	**int**	ID of the file in which the page is located.
page_id	**bigint**	ID of the page in the file.
error_type	**int**	Type of the error. The values can be: -1 = All hardware 823 errors 1 = 824 errors other than a bad checksum or a torn page (such as a bad page ID) 2 = Bad checksum 3 = Torn page
page_status	**int**	The status of the page-repair attempt: 2 = Queued for request from partner. 3 = Request sent to partner. 4 = Queued for automatic page repair (response received from partner). 5 = Automatic page repair succeeded and the page should be usable. 6 = Irreparable. This indicates that an error occurred during page-repair attempt, for example, because the page is also corrupted on the partner, the partner is disconnected, or a network problem occurred. This state is not terminal; if corruption is encountered again on the page, the page will be requested again from the partner.

COLUMN NAME	DATA TYPE	DESCRIPTION
modification_time	**datetime**	Time of last change to the page status.

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Automatic Page Repair (Availability Groups: Database Mirroring)
Dynamic Management Views and Functions (Transact-SQL)
suspect_pages (Transact-SQL)
Manage the suspect_pages Table (SQL Server)

Database Mirroring - sys.dm_db_mirroring_connections

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each connection established for database mirroring.

COLUMN NAME	DATA TYPE	DESCRIPTION
connection_id	**uniqueidentifier**	Identifier of the connection.
transport_stream_id	**uniqueidentifier**	Identifier of the SQL Server Network Interface (SNI) connection used by this connection for TCP/IP communications.
state	**smallint**	Current state of the connection. Possible values: 1 = NEW 2 = CONNECTING 3 = CONNECTED 4 = LOGGED_IN 5 = CLOSED
state_desc	**nvarchar(60)**	Current state of the connection. Possible values: NEW CONNECTING CONNECTED LOGGED_IN CLOSED
connect_time	**datetime**	Date and time at which the connection was opened.
login_time	**datetime**	Date and time at which login for the connection succeeded.
authentication_method	**nvarchar(128)**	Name of the Windows Authentication method, such as NTLM or KERBEROS. The value comes from Windows.

COLUMN NAME	DATA TYPE	DESCRIPTION
principal_name	**nvarchar(128)**	Name of the login that was validated for connection permissions. For Windows Authentication, this value is the remote user name. For certificate authentication, this value is the certificate owner.
remote_user_name	**nvarchar(128)**	Name of the peer user from the other database that is used by Windows Authentication.
last_activity_time	**datetime**	Date and time at which the connection was last used to send or receive information.
is_accept	**bit**	Indicates whether the connection originated on the remote side. 1 = The connection is a request accepted from the remote instance. 0 = The connection was started by the local instance.
login_state	**smallint**	State of the login process for this connection. Possible values: 0 = INITIAL 1 = WAIT LOGIN NEGOTIATE 2 = ONE ISC 3 = ONE ASC 4 = TWO ISC 5 = TWO ASC 6 = WAIT ISC Confirm 7 = WAIT ASC Confirm 8 = WAIT REJECT 9 = WAIT PRE-MASTER SECRET 10 = WAIT VALIDATION 11 = WAIT ARBITRATION 12 = ONLINE 13 = ERROR

COLUMN NAME	DATA TYPE	DESCRIPTION
login_state_desc	**nvarchar(60)**	Current state of login from the remote computer. Possible values: Connection handshake is initializing. Connection handshake is waiting for Login Negotiate message. Connection handshake has initialized and sent security context for authentication. Connection handshake has received and accepted security context for authentication. Connection handshake has initialized and sent security context for authentication. There is an optional mechanism available for authenticating the peers. Connection handshake has received and sent accepted security context for authentication. There is an optional mechanism available for authenticating the peers. Connection handshake is waiting for Initialize Security Context Confirmation message. Connection handshake is waiting for Accept Security Context Confirmation message. Connection handshake is waiting for SSPI rejection message for failed authentication. Connection handshake is waiting for Pre-Master Secret message. Connection handshake is waiting for Validation message. Connection handshake is waiting for Arbitration message. Connection handshake is complete and is online (ready) for message exchange. Connection is in error.
peer_certificate_id	**int**	The local object ID of the certificate used by the remote instance for authentication. The owner of this certificate must have CONNECT permissions to the database mirroring endpoint.
encryption_algorithm	**smallint**	Encryption algorithm that is used for

COLUMN NAME	DATA TYPE	DESCRIPTION
encryption_algorithm	smallint	Encryption algorithm that is used for this connection. NULLABLE. Possible values: **Value:**0 **Description:** None **DDL Option:** Disabled **Value:**1 **Description:** RC4 **DDL Option:** {Required | Required algorithm RC4} **Value:**2 **Description:** AES **DDL Option:** Required algorithm AES **Value:**3 **Description:** None, RC4 **DDL Option:** {Supported | Supported algorithm RC4} **Value:**4 **Description:** none, AES **DDL Option:** Supported algorithm RC4 **Value:**5 **Description:** RC4, AES **DDL Option:** Required algorithm RC4 AES **Value:**6 **Description:** AES, RC4 **DDL Option:** Required Algorithm AES RC4 **Value:**7 **Description:** NONE, RC4, AES **DDL Option:** Supported Algorithm RC4 AES **Value:**8 **Description:** NONE, AES, RC4 **DDL Option:** Supported algorithm AES RC4 **Note:** The RC4 algorithm is only supported for backward compatibility.

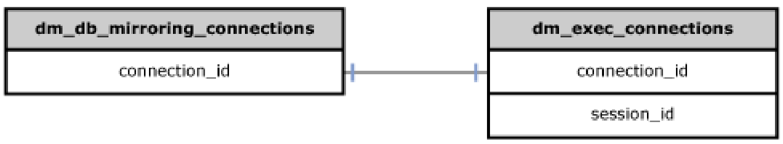
COLUMN NAME	DATA TYPE	DESCRIPTION
		New material can only be encrypted using RC4 or RC4_128 when the database is in compatibility level 90 or 100. (Not recommended.) Use a newer algorithm such as one of the AES algorithms instead. In SQL Server 2012 (11.x) and higher versions, material encrypted using RC4 or RC4_128 can be decrypted in any compatibility level.
encryption_algorithm_desc	**nvarchar(60)**	Textual representation of the encryption algorithm. NULLABLE. Possible Values: **Description:** None **DDL Option:** Disabled **Description:** RC4 **DDL Option:** {Required | Required Algorithm RC4} **Description:** AES **DDL Option:** Required Algorithm AES **Description:** NONE, RC4 **DDL Option:** {Supported | Supported Algorithm RC4} **Description:** NONE, AES **DDL Option:** Supported Algorithm RC4 **Description:** RC4, AES **DDL Option:** Required Algorithm RC4 AES **Description:** AES, RC4 **DDL Option:** Required Algorithm AES RC4 **Description:** NONE, RC4, AES **DDL Option:** Supported Algorithm RC4 AES **Description:** NONE, AES, RC4 **DDL Option:** Supported Algorithm AES RC4
receives_posted	**smallint**	Number of asynchronous network receives that have not yet completed for this connection.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_receive_flow_controlled	**bit**	Whether network receives have been postponed due to flow control because the network is busy. 1 = True
sends_posted	**smallint**	The number of asynchronous network sends that have not yet completed for this connection.
is_send_flow_controlled	**bit**	Whether network sends have been postponed due to network flow control because the network is busy. 1 = True
total_bytes_sent	**bigint**	Total number of bytes sent by this connection.
total_bytes_received	**bigint**	Total number of bytes received by this connection.
total_fragments_sent	**bigint**	Total number of database mirroring message fragments sent by this connection.
total_fragments_received	**bigint**	Total number of database mirroring message fragments received by this connection.
total_sends	**bigint**	Total number of network send requests issued by this connection.
total_receives	**bigint**	Total number of network receive requests issued by this connection.
peer_arbitration_id	**uniqueidentifier**	Internal identifier for the endpoint. NULLABLE.

Permissions

Requires VIEW SERVER STATE permission on the server.

Physical Joins



Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_db_mirroring_connections.connection_id	**dm_exec_connections.connection_id**	One-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)
Monitoring Database Mirroring (SQL Server)

Execution Related Dynamic Management Views and Functions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management objects:

sys.dm_exec_background_job_queue	sys.dm_exec_background_job_queue_stats
sys.dm_exec_cached_plan_dependent_objects	sys.dm_exec_cached_plans
sys.dm_exec_compute_node_errors	sys.dm_exec_compute_node_status
sys.dm_exec_compute_nodes	sys.dm_exec_connections
sys.dm_exec_cursors	sys.dm_exec_describe_first_result_set
sys.dm_exec_describe_first_result_set_for_object	sys.dm_exec_distributed_request_steps
sys.dm_exec_distributed_requests	sys.dm_exec_distributed_sql_requests
sys.dm_exec_dms_services	sys.dm_exec_dms_workers
sys.dm_exec_external_operations	sys.dm_exec_external_work
sys.dm_exec_function_stats	sys.dm_exec_input_buffer
sys.dm_exec_plan_attributes	sys.dm_exec_procedure_stats
sys.dm_exec_query_memory_grants	sys.dm_exec_query_optimizer_info
sys.dm_exec_query_optimizer_memory_gateways	sys.dm_exec_query_plan
sys.dm_exec_query_parallel_workers	sys.dm_exec_query_profiles
sys.dm_exec_query_resource_semaphores	sys.dm_exec_query_statistics_xml
sys.dm_exec_query_stats	sys.dm_exec_requests
sys.dm_exec_session_wait_stats	sys.dm_exec_sessions
sys.dm_exec_sql_text	sys.dm_exec_text_query_plan
sys.dm_exec_trigger_stats	sys.dm_exec_valid_use_hints

sys.dm_exec_xml_handles	sys.dm_external_script_execution_stats
sys.dm_external_script_requests	

NOTE

The **sys.dm_exec_query_transformation_stats** dynamic management view is identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

See Also

Dynamic Management Views and Functions (Transact-SQL)
System Views (Transact-SQL)

sys.dm_exec_background_job_queue (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each query processor job that is scheduled for asynchronous (background) execution.

> **NOTE!!** To call this from **Azure SQL Data Warehouse** or **Parallel Data Warehouse**, use the name **sys.dm_pdw_nodes_exec_background_job_queue**.

COLUMN NAME	DATA TYPE	DESCRIPTION
time_queued	**datetime**	Time when the job was added to the queue.
job_id	**int**	Job identifier.
database_id	**int**	Database on which the job is to execute.
object_id1	**int**	Value depends on the job type. For more information, see the Remarks section.
object_id2	**int**	Value depends on the job type. For more information, see the Remarks section.
object_id3	**int**	Value depends on the job type. For more information, see the Remarks section.
object_id4	**int**	Value depends on the job type. For more information, see the Remarks section.
error_code	**int**	Error code if the job reinserted due to failure. NULL if suspended, not picked up, or completed.
request_type	**smallint**	Type of the job request.
retry_count	**smallint**	Number of times the job was picked from the queue and reinserted because of lack of resources or other reasons.
in_progress	**smallint**	Indicates whether the job has started execution. 1 = Started 0 = Still waiting

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**smallint**	Session identifier.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

This view returns information only for asynchronous update statistics jobs. For more information about asynchronous update statistics, see Statistics.

The values of **object_id1** through **object_id4** depend on the type of the job request. The following table summarizes the meaning of these columns for the different job types.

REQUEST TYPE	OBJECT_ID1	OBJECT_ID2	OBJECT_ID3	OBJECT_ID4
Asynchronous update statistics	Table or view ID	Statistics ID	Not used	Not used

Examples

The following example returns the number of active asynchronous jobs in the background queue for each database in the instance of SQL Server.

```
SELECT DB_NAME(database_id) AS [Database], COUNT(*) AS [Active Async Jobs]
FROM sys.dm_exec_background_job_queue
WHERE in_progress = 1
GROUP BY database_id;
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)
Statistics
KILL STATS JOB (Transact-SQL)

sys.dm_exec_background_job_queue_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row that provides aggregate statistics for each query processor job submitted for asynchronous (background) execution.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_exec_background_job_queue_stats**.

COLUMN NAME	DATA TYPE	DESCRIPTION
queue_max_len	**int**	Maximum length of the queue.
enqueued_count	**int**	Number of requests successfully posted to the queue.
started_count	**int**	Number of requests that started execution.
ended_count	**int**	Number of requests serviced to either success or failure.
failed_lock_count	**int**	Number of requests that failed due to lock contention or deadlock.
failed_other_count	**int**	Number of requests that failed due to other reasons.
failed_giveup_count	**int**	Number of requests that failed because retry limit has been reached.
enqueue_failed_full_count	**int**	Number of failed enqueue attempts because the queue is full.
enqueue_failed_duplicate_count	**int**	Number of duplicate enqueue attempts.
elapsed_avg_ms	**int**	Average elapsed time of request in milliseconds.
elapsed_max_ms	**int**	Elapsed time of the longest request in milliseconds.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Remarks

This view returns information only for asynchronous update statistics jobs. For more information about asynchronous update statistics, see Statistics.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

A. Determining the percentage of failed background jobs

The following example returns the percentage of failed background jobs for all executed queries.

```
SELECT
        CASE ended_count WHEN 0
                THEN 'No jobs ended'
                ELSE CAST((failed_lock_count + failed_giveup_count + failed_other_count) / CAST(ended_count AS
float) * 100 AS varchar(20))
        END AS [Percent Failed]
FROM sys.dm_exec_background_job_queue_stats;
GO
```

B. Determining the percentage of failed enqueue attempts

The following example returns the percentage of failed enqueue attempts for all executed queries.

```
SELECT
        CASE enqueued_count WHEN 0
                THEN 'No jobs posted'
                ELSE CAST((enqueue_failed_full_count + enqueue_failed_duplicate_count) / CAST(enqueued_count
AS float) * 100 AS varchar(20))
        END AS [Percent Enqueue Failed]
FROM sys.dm_exec_background_job_queue_stats;
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_exec_cached_plan_dependent_objects (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each Transact-SQL execution plan, common language runtime (CLR) execution plan, and cursor associated with a plan.

Syntax

```
dm_exec_cached_plan_dependent_objects(plan_handle)
```

Arguments

plan_handle
Uniquely identifies a query execution plan for a batch that has executed and its plan resides in the plan cache. *plan_handle* is **varbinary(64)**. The *plan_handle* can be obtained from the following dynamic management objects:

- sys.dm_exec_cached_plans (Transact-SQL)
- sys.dm_exec_query_stats (Transact-SQL)
- sys.dm_exec_requests (Transact-SQL)

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
usecounts	**int**	Number of times the execution context or cursor has been used. Column is not nullable.
memory_object_address	**varbinary(8)**	Memory address of the execution context or cursor. Column is not nullable.
cacheobjtype	**nvarchar(50)**	The Plan cache object type. Column is not nullable. Possible values are Executable plan CLR compiled function CLR compiled procedure Cursor

Permissions

Requires VIEW SERVER STATE permission on the server.

Physical Joins

Relationship Cardinalities

FROM	TO	ON	RELATIONSHIP
dm_exec_cached_plan_dependent_objects	**dm_os_memory_objects**	**memory_object_address**	One-to-one

See Also

Execution Related Dynamic Management Views and Functions (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)
sys.syscacheobjects (Transact-SQL)

sys.dm_exec_cached_plans (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each query plan that is cached by SQL Server for faster query execution. You can use this dynamic management view to find cached query plans, cached query text, the amount of memory taken by cached plans, and the reuse count of the cached plans.

In Azure SQL Database, dynamic management views cannot expose information that would impact database containment or expose information about other databases the user has access to. To avoid exposing this information, every row that contains data that doesn't belong to the connected tenant is filtered out. In addition, the values in the columns **memory_object_address** and **pool_id** are filtered; the column value is set to NULL.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_exec_cached_plans**.

COLUMN NAME	DATA TYPE	DESCRIPTION
bucketid	**int**	ID of the hash bucket in which the entry is cached. The value indicates a range from 0 through the hash table size for the type of cache. For the SQL Plans and Object Plans caches, the hash table size can be up to 10007 on 32-bit systems and up to 40009 on 64-bit systems. For the Bound Trees cache, the hash table size can be up to 1009 on 32-bit systems and up to 4001 on 64-bit systems. For the Extended Stored Procedures cache the hash table size can be up to 127 on 32-bit and 64-bit systems.
refcounts	**int**	Number of cache objects that are referencing this cache object. **Refcounts** must be at least 1 for an entry to be in the cache.
usecounts	**int**	Number of times the cache object has been looked up. Not incremented when parameterized queries find a plan in the cache. Can be incremented multiple times when using showplan.
size_in_bytes	**int**	Number of bytes consumed by the cache object.

COLUMN NAME	DATA TYPE	DESCRIPTION
memory_object_address	**varbinary(8)**	Memory address of the cached entry. This value can be used with sys.dm_os_memory_objects to get the memory breakdown of the cached plan and with sys.dm_os_memory_cache_entries_entries to obtain the cost of caching the entry.
cacheobjtype	**nvarchar(34)**	Type of object in the cache. The value can be one of the following: Compiled Plan Compiled Plan Stub Parse Tree Extended Proc CLR Compiled Func CLR Compiled Proc
objtype	**nvarchar(16)**	Type of object. Below are the possible values and their corresponding descriptions. Proc: Stored procedure Prepared: Prepared statement Adhoc: Ad hoc query. Refers to Transact-SQL submitted as language events by using **osql** or **sqlcmd** instead of as remote procedure calls. ReplProc: Replication-filter-procedure Trigger: Trigger View: View Default: Default UsrTab: User table SysTab: System table Check: CHECK constraint Rule: Rule
plan_handle	**varbinary(64)**	Identifier for the in-memory plan. This identifier is transient and remains constant only while the plan remains in the cache. This value may be used with the following dynamic management functions: sys.dm_exec_sql_text sys.dm_exec_query_plan sys.dm_exec_plan_attributes
pool_id	**int**	The ID of the resource pool against which this plan memory usage is accounted for.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

1

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

A. Returning the batch text of cached entries that are reused

The following example returns the SQL text of all cached entries that have been used more than once.

```
SELECT usecounts, cacheobjtype, objtype, text
FROM sys.dm_exec_cached_plans
CROSS APPLY sys.dm_exec_sql_text(plan_handle)
WHERE usecounts > 1
ORDER BY usecounts DESC;
GO
```

B. Returning query plans for all cached triggers

The following example returns the query plans of all cached triggers.

```
SELECT plan_handle, query_plan, objtype
FROM sys.dm_exec_cached_plans
CROSS APPLY sys.dm_exec_query_plan(plan_handle)
WHERE objtype ='Trigger';
GO
```

C. Returning the SET options with which the plan was compiled

The following example returns the SET options with which the plan was compiled. The `sql_handle` for the plan is also returned. The PIVOT operator is used to output the `set_options` and `sql_handle` attributes as columns rather than as rows. For more information about the value returned in `set_options`, see sys.dm_exec_plan_attributes (Transact-SQL).

```
SELECT plan_handle, pvt.set_options, pvt.sql_handle
FROM (
      SELECT plan_handle, epa.attribute, epa.value
      FROM sys.dm_exec_cached_plans
      OUTER APPLY sys.dm_exec_plan_attributes(plan_handle) AS epa
      WHERE cacheobjtype = 'Compiled Plan'
      ) AS ecpa
PIVOT (MAX(ecpa.value) FOR ecpa.attribute IN ("set_options", "sql_handle")) AS pvt;
GO
```

D. Returning the memory breakdown of all cached compiled plans

The following example returns a breakdown of the memory used by all compiled plans in the cache.

```
SELECT plan_handle, ecp.memory_object_address AS CompiledPlan_MemoryObject,
    omo.memory_object_address, type, page_size_in_bytes
FROM sys.dm_exec_cached_plans AS ecp
JOIN sys.dm_os_memory_objects AS omo
    ON ecp.memory_object_address = omo.memory_object_address
    OR ecp.memory_object_address = omo.parent_address
WHERE cacheobjtype = 'Compiled Plan';
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_query_plan (Transact-SQL)
sys.dm_exec_plan_attributes (Transact-SQL)
sys.dm_exec_sql_text (Transact-SQL)
sys.dm_os_memory_objects (Transact-SQL)
sys.dm_os_memory_cache_entries (Transact-SQL)
FROM (Transact-SQL)

sys.dm_exec_compute_node_errors (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns errors that occur on PolyBase compute nodes.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
error_id	**nvarchar(36)**	Unique numeric id associated with the error .	Unique across all query errors in the system
source	**nvarchar(255)**	Source thread or process description	
type	**nvarchar(255)**	Type of error.	
create_time	**datetime**	The time of the error occurrence	
compute_node_id	**int**	Identifier of the specific compute node	See compute_node_id of sys.dm_exec_compute_nodes (Transact-SQL)
rexecution_id	**nvarchar(36)**	Identifier of the PolyBase query, if any.	
spid	**int**	Identifier of the SQL Server session	
thread_id	**int**	Numeric identifier of the thread on which the error occurred.	
details	nvarchar(4000)	Full description of the details of the error.	

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_compute_node_status (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds additional information about the performance and status of all PolyBase nodes. Lists one row per node.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
compute_node_id	**int**	Unique numeric id associated with the node.	Unique across scale-out cluster regardless of type.
process_id	**int**		
process_name	**nvarchar(255)**	Logical name of the node.	Any string of appropriate length.
allocated_memory	**bigint**	Total allocated memory on this node.	
available_memory	**bigint**	Total available memory on this node.	
process_cpu_usage	**bigint**	Total process CPU usage, in ticks.	
total_cpu_usage	**bigint**	Total CPU usage, in ticks.	
thread_count	**bigint**	Total number of threads in use on this node.	
handle_count	**bigint**	Total number of handles in use on this node.	
total_elapsed_time	**bigint**	Total time elapsed since system start or restart.	Total time elapsed since system start or restart. If total_elapsed_time exceeds the maximum value for an integer (24.8 days in milliseconds), it will cause materialization failure due to overflow.The maximum value in milliseconds is equivalent to 24.8 days.
is_available	**bit**	Flag indicating whether this node is available.	
sent_time	**datetime**	Last time a network package was sent by this	

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
received_time	**datetime**	Last time a network package was sent by this node.	
error_id	**nvarchar(36)**	Unique identifier of the last error that occurred on this node.	

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_compute_nodes (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about nodes used with PolyBase data management. It lists one row per node.

Use this DMV to see the list of all nodes in the scale-out cluster with their role, name and IP address.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
compute_node_id	**int**	Unique numeric id associated with the node. Key for this view.	Unique across scale-out cluster regardless of type.
type	**nvarchar(32)**	Type of the node.	'COMPUTE', 'HEAD'
name	**nvarchar(32)**	Logical name of the node.	Any string of appropriate length.
address	**nvarchar(32)**	P address of this node.	IP address range

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_connections (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the connections established to this instance of SQL Server and the details of each connection. Returns server wide connection information for SQL Server. Returns current database connection information for SQL Database.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use sys.dm_pdw_exec_connections (Transact-SQL).

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**int**	Identifies the session associated with this connection. Is nullable.
most_recent_session_id	**int**	Represents the session ID for the most recent request associated with this connection. (SOAP connections can be reused by another session.) Is nullable.
connect_time	**datetime**	Timestamp when connection was established. Is not nullable.
net_transport	**nvarchar(40)**	Always returns **Session** when a connection has multiple active result sets (MARS) enabled. **Note:** Describes the physical transport protocol that is used by this connection. Is not nullable.
protocol_type	**nvarchar(40)**	Specifies the protocol type of the payload. It currently distinguishes between TDS (TSQL) and SOAP. Is nullable.
protocol_version	**int**	Version of the data access protocol associated with this connection. Is nullable.
endpoint_id	**int**	An identifier that describes what type of connection it is. This endpoint_id can be used to query the sys.endpoints view. Is nullable.
encrypt_option	**nvarchar(40)**	Boolean value to describe whether encryption is enabled for this connection. Is not nullable.

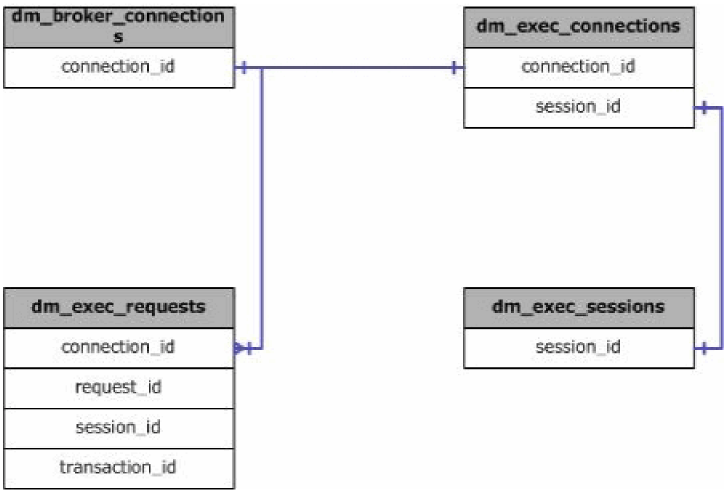
COLUMN NAME	DATA TYPE	DESCRIPTION
auth_scheme	**nvarchar(40)**	Specifies SQL Server/Windows Authentication scheme used with this connection. Is not nullable.
node_affinity	**smallint**	Identifies the memory node to which this connection has affinity. Is not nullable.
num_reads	**int**	Number of byte reads that have occurred over this connection. Is nullable.
num_writes	**int**	Number of byte writes that have occurred over this connection. Is nullable.
last_read	**datetime**	Timestamp when last read occurred over this connection. Is nullable.
last_write	**datetime**	Timestamp when last write occurred over this connection. Not Is nullable.
net_packet_size	**int**	Network packet size used for information and data transfer. Is nullable.
client_net_address	**varchar(48)**	Host address of the client connecting to this server. Is nullable. Prior to V12 in Azure SQL Database, this column always returns NULL.
client_tcp_port	**int**	Port number on the client computer that is associated with this connection. Is nullable. In Azure SQL Database, this column always returns NULL.
local_net_address	**varchar(48)**	Represents the IP address on the server that this connection targeted. Available only for connections using the TCP transport provider. Is nullable. In Azure SQL Database, this column always returns NULL.
local_tcp_port	**int**	Represents the server TCP port that this connection targeted if it were a connection using the TCP transport. Is nullable. In Azure SQL Database, this column always returns NULL.

COLUMN NAME	DATA TYPE	DESCRIPTION
connection_id	**uniqueidentifier**	Identifies each connection uniquely. Is not nullable.
parent_connection_id	**uniqueidentifier**	Identifies the primary connection that the MARS session is using. Is nullable.
most_recent_sql_handle	**varbinary(64)**	The SQL handle of the last request executed on this connection. The most_recent_sql_handle column is always in sync with the most_recent_session_id column. Is nullable.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Physical Joins



Relationship Cardinalities

dm_exec_sessions.session_id	dm_exec_connections.session_id	One-to-one
dm_exec_requests.connection_id	dm_exec_connections.connection_id	Many to one
dm_broker_connections.connection_id	dm_exec_connections.connection_id	One to one

Examples

Typical query to gather information about a queries own connection.

```
SELECT
    c.session_id, c.net_transport, c.encrypt_option,
    c.auth_scheme, s.host_name, s.program_name,
    s.client_interface_name, s.login_name, s.nt_domain,
    s.nt_user_name, s.original_login_name, c.connect_time,
    s.login_time
FROM sys.dm_exec_connections AS c
JOIN sys.dm_exec_sessions AS s
    ON c.session_id = s.session_id
WHERE c.session_id = @@SPID;
```

See Also

Execution Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_exec_cursors (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the cursors that are open in various databases.

Syntax

```
dm_exec_cursors (session_id | 0 )
```

Arguments

session_id | 0
ID of the session. If *session_id* is specified, this function returns information about cursors in the specified session.

If 0 is specified, the function returns information about all cursors for all sessions.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**int**	ID of the session that holds this cursor.
cursor_id	**int**	ID of the cursor object.
name	**nvarchar(256)**	Name of the cursor as defined by the user.
properties	**nvarchar(256)**	Specifies the properties of the cursor. The values of the following properties are concatenated to form the value of this column: Declaration Interface Cursor Type Cursor Concurrency Cursor scope Cursor nesting level For example, the value returned in this column might be "TSQL | Dynamic | Optimistic | Global (0)".
sql_handle	**varbinary(64)**	Handle to the text of the batch that declared the cursor.

COLUMN NAME	DATA TYPE	DESCRIPTION
statement_start_offset	**int**	Number of characters into the currently executing batch or stored procedure at which the currently executing statement starts. Can be used together with the **sql_handle**, the **statement_end_offset**, and the sys.dm_exec_sql_text dynamic management function to retrieve the currently executing statement for the request.
statement_end_offset	**int**	Number of characters into the currently executing batch or stored procedure at which the currently executing statement ends. Can be used together with the **sql_handle**, the **statement_start_offset**, and the **sys.dm_exec_sql_text** dynamic management function to retrieve the currently executing statement for the request.
plan_generation_num	**bigint**	A sequence number that can be used to distinguish between instances of plans after recompilation.
creation_time	**datetime**	Timestamp when this cursor was created.
is_open	**bit**	Specifies whether the cursor is open.
is_async_population	**bit**	Specifies whether the background thread is still asynchronously populating a KEYSET or STATIC cursor.
is_close_on_commit	**bit**	Specifies whether the cursor was declared by using CURSOR_CLOSE_ON_COMMIT. 1 = Cursor will be closed when the transaction ends.
fetch_status	**int**	Returns last fetch status of the cursor. This is the last returned @@FETCH_STATUS value.
fetch_buffer_size	**int**	Returns information about the size of the fetch buffer. 1 = Transact-SQL cursors. This can be set to a higher value for API cursors.

COLUMN NAME	DATA TYPE	DESCRIPTION
fetch_buffer_start	**int**	For FAST_FORWARD and DYNAMIC cursors, it returns 0 if the cursor is not open or if it is positioned before the first row. Otherwise, it returns -1. For STATIC and KEYSET cursors, it returns 0 if the cursor is not open, and -1 if the cursor is positioned beyond the last row. Otherwise, it returns the row number in which it is positioned.
ansi_position	**int**	Cursor position within the fetch buffer.
worker_time	**bigint**	Time spent, in microseconds, by the workers executing this cursor.
reads	**bigint**	Number of reads performed by the cursor.
writes	**bigint**	Number of writes performed by the cursor.
dormant_duration	**bigint**	Milliseconds since the last query (open or fetch) on this cursor was started.

Permissions

Requires VIEW SERVER STATE permission on the server.

Remarks

The following table provides information about the cursor declaration interface and includes the possible values for the properties column.

PROPERTY	DESCRIPTION
API	Cursor was declared by using one of the data access APIs (ODBC, OLEDB).
TSQL	Cursor was declared by using the Transact-SQL DECLARE CURSOR syntax.

The following table provides information about the cursor type and includes the possible values for the properties column.

TYPE	DESCRIPTION
Keyset	Cursor was declared as Keyset.
Dynamic	Cursor was declared as Dynamic.

TYPE	DESCRIPTION
Snapshot	Cursor was declared as Snapshot or Static.
Fast_Forward	Cursor was declared as Fast Forward.

The following table provides information about cursor concurrency and includes the possible values for the properties column.

CONCURRENCY	DESCRIPTION
Read Only	Cursor was declared as read-only.
Scroll Locks	Cursor uses scroll locks.
Optimistic	Cursor uses optimistic concurrency control.

The following table provides information about cursor scope and includes the possible values for the properties column.

SCOPE	DESCRIPTION
Local	Specifies that the scope of the cursor is local to the batch, stored procedure, or trigger in which the cursor was created.
Global	Specifies that the scope of the cursor is global to the connection.

Examples

A. Detecting old cursors

This example returns information about cursors that have been open on the server longer than the specified time of 36 hours.

```
SELECT creation_time, cursor_id, name, c.session_id, login_name
FROM sys.dm_exec_cursors(0) AS c
JOIN sys.dm_exec_sessions AS s ON c.session_id = s.session_id
WHERE DATEDIFF(hh, c.creation_time, GETDATE()) > 36;
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_sessions (Transact-SQL)

sys.dm_exec_describe_first_result_set (Transact-SQL)

5/4/2018 • 9 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This dynamic management function takes a Transact-SQL statement as a parameter and describes the metadata of the first result set for the statement.

sys.dm_exec_describe_first_result_set has the same result set definition as sys.dm_exec_describe_first_result_set_for_object (Transact-SQL) and is similar to sp_describe_first_result_set (Transact-SQL).

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_exec_describe_first_result_set(@tsql, @params, @include_browse_information)
```

Arguments

@tsql
One or more Transact-SQL statements. *Transact-SQL_batch* may be **nvarchar(*n*)** or **nvarchar(max)**.

@params
@params provides a declaration string for parameters for the Transact-SQL batch, similar to sp_executesql. Parameters may be **nvarchar(n)** or **nvarchar(max)**.

Is one string that contains the definitions of all parameters that have been embedded in the Transact-SQL_*batch*. The string must be either a Unicode constant or a Unicode variable. Each parameter definition consists of a parameter name and a data type. *n* is a placeholder that indicates additional parameter definitions. Every parameter specified in stmt must be defined in @params. If the Transact-SQL statement or batch in the statement does not contain parameters, @params is not required. NULL is the default value for this parameter.

@include_browse_information
If set to 1, each query is analyzed as if it has a FOR BROWSE option on the query. Additional key columns and source table information are returned.

Table Returned

This common metadata is returned as a result set. One row for each column in the results metadata describes the type and nullability of the column in the format shown in the following table. If the first statement does not exist for every control path, a result set with zero rows is returned.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_hidden	**bit**	Specifies that the column is an extra column added for browsing and informational purposes that does not actually appear in the result set.

COLUMN NAME	DATA TYPE	DESCRIPTION
column_ordinal	**int**	Contains the ordinal position of the column in the result set. Position of the first column will be specified as 1.
name	**sysname**	Contains the name of the column if a name can be determined. If not, will contain NULL.
is_nullable	**bit**	Contains the following values: Value 1 if column allows NULLs. Value 0 if the column does not allow NULLs. Value 1 if it cannot be determined that the column allows NULLs.
system_type_id	**int**	Contains the system_type_id of the column data type as specified in sys.types. For CLR types, even though the system_type_name column will return NULL, this column will return the value 240.
system_type_name	**nvarchar(256)**	Contains the name and arguments (such as length, precision, scale), specified for the data type of the column. If data type is a user-defined alias type, the underlying system type is specified here. If data type is a CLR user-defined type, NULL is returned in this column.
max_length	**smallint**	Maximum length (in bytes) of the column. -1 = Column data type is **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, or **xml**. For **text** columns, the **max_length** value will be 16 or the value set by **sp_tableoption 'text in row'**.
precision	**tinyint**	Precision of the column if numeric-based. Otherwise returns 0.
scale	**tinyint**	Scale of column if numeric-based. Otherwise returns 0.
collation_name	**sysname**	Name of the collation of the column if character-based. Otherwise returns NULL.

COLUMN NAME	DATA TYPE	DESCRIPTION
user_type_id	**int**	For CLR and alias types, contains the user_type_id of the data type of the column as specified in sys.types. Otherwise is NULL.
user_type_database	**sysname**	For CLR and alias types, contains the name of the database in which the type is defined. Otherwise is NULL.
user_type_schema	**sysname**	For CLR and alias types, contains the name of the schema in which the type is defined. Otherwise is NULL.
user_type_name	**sysname**	For CLR and alias types, contains the name of the type. Otherwise is NULL.
assembly_qualified_type_name	**nvarchar(4000)**	For CLR types, returns the name of the assembly and class defining the type. Otherwise is NULL.
xml_collection_id	**int**	Contains the xml_collection_id of the data type of the column as specified in sys.columns. This column returns NULL if the type returned is not associated with an XML schema collection.
xml_collection_database	**sysname**	Contains the database in which the XML schema collection associated with this type is defined. This column returns NULL if the type returned is not associated with an XML schema collection.
xml_collection_schema	**sysname**	Contains the schema in which the XML schema collection associated with this type is defined. This column returns NULL if the type returned is not associated with an XML schema collection.
xml_collection_name	**sysname**	Contains the name of the XML schema collection associated with this type. This column returns NULL if the type returned is not associated with an XML schema collection.
is_xml_document	**bit**	Returns 1 if the returned data type is XML and that type is guaranteed to be a complete XML document (including a root node), as opposed to an XML fragment). Otherwise returns 0.
is_case_sensitive	**bit**	Returns 1 if the column is of a case-sensitive string type. Returns 0 if it is not.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_fixed_length_clr_type	**bit**	Returns 1 if the column is of a fixed-length CLR type. Returns 0 if it is not.
source_server	**sysname**	Name of the originating server (if it originates from a remote server). The name is given as it appears in sys.servers. Returns NULL if the column originates on the local server or if it cannot be determined which server it originates on. Is only populated if browsing information is requested.
source_database	**sysname**	Name of the originating database returned by the column in this result. Returns NULL if the database cannot be determined. Is only populated if browsing information is requested.
source_schema	**sysname**	Name of the originating schema returned by the column in this result. Returns NULL if the schema cannot be determined. Is only populated if browsing information is requested.
source_table	**sysname**	Name of the originating table returned by the column in this result. Returns NULL if the table cannot be determined. Is only populated if browsing information is requested.
source_column	**sysname**	Name of the originating column returned by the result column. Returns NULL if the column cannot be determined. Is only populated if browsing information is requested.
is_identity_column	**bit**	Returns 1 if the column is an identity column and 0 if not. Returns NULL if it cannot be determined that the column is an identity column.
is_part_of_unique_key	**bit**	Returns 1 if the column is part of a unique index (including unique and primary constraints) and 0 if it is not. Returns NULL if it cannot be determined that the column is part of a unique index. Is only populated if browsing information is requested.
is_updateable	**bit**	Returns 1 if the column is updateable and 0 if not. Returns NULL if it cannot be determined that the column is updateable.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_computed_column	**bit**	Returns 1 if the column is a computed column and 0 if not. Returns NULL if it cannot be determined if the column is a computed column.
is_sparse_column_set	**bit**	Returns 1 if the column is a sparse column and 0 if not. Returns NULL if it cannot be determined that the column is a part of a sparse column set.
ordinal_in_order_by_list	**smallint**	ihe position of this column is in ORDER BY list. Returns NULL if the column does not appear in the ORDER BY list, or if the ORDER BY list cannot be uniquely determined.
order_by_list_length	**smallint**	The length of the ORDER BY list. NULL is returned if there is no ORDER BY list or if the ORDER BY list cannot be uniquely determined. Note that this value will be the same for all rows returned by sp_describe_first_result_set.
order_by_is_descending	**smallint NULL**	If the ordinal_in_order_by_list is not NULL, the **order_by_is_descending** column reports the direction of the ORDER BY clause for this column. Otherwise it reports NULL.
error_number	**int**	Contains the error number returned by the function. If no error occurred, the column will contain NULL.
error_severity	**int**	Contains the severity returned by the function. If no error occurred, the column will contain NULL.
error_state	**int**	Contains the state message. returned by the function. If no error occurred, the column will contain NULL.
error_message	**nvarchar(4096)**	Contains the message returned by the function. If no error occurred, the column will contain NULL.
error_type	**int**	Contains an integer representing the error being returned. Maps to error_type_desc. See the list under remarks.
error_type_desc	**nvarchar(60)**	Contains a short uppercase string representing the error being returned. Maps to error_type. See the list under remarks.

Remarks

This function uses the same algorithm as **sp_describe_first_result_set**. For more information, see sp_describe_first_result_set (Transact-SQL).

The following table lists the error types and their descriptions

ERROR_TYPE	ERROR_TYPE	DESCRIPTION
1	MISC	All errors that are not otherwise described.
2	SYNTAX	A syntax error occurred in the batch.
3	CONFLICTING_RESULTS	The result could not be determined because of a conflict between two possible first statements.
4	DYNAMIC_SQL	The result could not be determined because of dynamic SQL that could potentially return the first result.
5	CLR_PROCEDURE	The result could not be determined because a CLR stored procedure could potentially return the first result.
6	CLR_TRIGGER	The result could not be determined because a CLR trigger could potentially return the first result.
7	EXTENDED_PROCEDURE	The result could not be determined because an extended stored procedure could potentially return the first result.
8	UNDECLARED_PARAMETER	The result could not be determined because the data type of one or more of the result set's columns potentially depends on an undeclared parameter.
9	RECURSION	The result could not be determined because the batch contains a recursive statement.
10	TEMPORARY_TABLE	The result could not be determined because the batch contains a temporary table and is not supported by **sp_describe_first_result_set** .
11	UNSUPPORTED_STATEMENT	The result could not be determined because the batch contains a statement that is not supported by **sp_describe_first_result_set** (e.g., FETCH, REVERT etc.).
12	OBJECT_TYPE_NOT_SUPPORTED	The @object_id passed to the function is not supported (i.e. not a stored procedure)
13	OBJECT_DOES_NOT_EXIST	The @object_id passed to the function was not found in the system catalog.

Permissions

Requires permission to execute the @tsql argument.

Examples

Additional examples in the topic sp_describe_first_result_set (Transact-SQL) can be adapted to use **sys.dm_exec_describe_first_result_set**.

A. Returning information about a single Transact-SQL statement

The following code returns information about the results of a Transact-SQL statement.

```
USE AdventureWorks2012;
GO
SELECT * FROM sys.dm_exec_describe_first_result_set
(N'SELECT object_id, name, type_desc FROM sys.indexes', null, 0) ;
```

B. Returning information about a procedure

The following example creates a stored procedure named pr_TestProc that returns two result sets. Then the example demonstrates that **sys.dm_exec_describe_first_result_set** returns information about the first result set in the procedure.

```
USE AdventureWorks2012;
GO

CREATE PROC Production.TestProc
AS
SELECT Name, ProductID, Color FROM Production.Product ;
SELECT Name, SafetyStockLevel, SellStartDate FROM Production.Product ;
GO

SELECT * FROM sys.dm_exec_describe_first_result_set
('Production.TestProc', NULL, 0) ;
```

C. Returning metadata from a batch that contains multiple statements

The following example evaluates a batch that contains two Transact-SQL statements. The result set describes the first result set returned.

```
USE AdventureWorks2012;
GO

SELECT * FROM sys.dm_exec_describe_first_result_set(
N'SELECT CustomerID, TerritoryID, AccountNumber FROM Sales.Customer WHERE CustomerID = @CustomerID;
SELECT * FROM Sales.SalesOrderHeader;',
N'@CustomerID int', 0) AS a;
GO
```

See Also

sp_describe_first_result_set (Transact-SQL)
sp_describe_undeclared_parameters (Transact-SQL)
sys.dm_exec_describe_first_result_set_for_object (Transact-SQL)

sys.dm_exec_describe_first_result_set_for_object (Transact-SQL)

5/4/2018 • 8 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This dynamic management function takes an @object_id as a parameter and describes the first result metadata for the module with that ID. The @object_id specified can be the ID of a Transact-SQL stored procedure or a Transact-SQL trigger. If it is the ID of any other object (such as a view, table, function, or CLR procedure), an error will be specified in the error columns of the result.

sys.dm_exec_describe_first_result_set_for_object has the same result set definition as sys.dm_exec_describe_first_result_set (Transact-SQL) and is similar to sp_describe_first_result_set (Transact-SQL).

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_exec_describe_first_result_set_for_object
    ( @object_id , @include_browse_information )
```

Arguments

@object_id
The @object_id of a Transact-SQL stored procedure or a Transact-SQL trigger. @object_id is type **int**.

@include_browse_information
@include_browse_information is type **bit**. If set to 1, each query is analyzed as if it has a FOR BROWSE option on the query. Returns additional key columns and source table information.

Table Returned

This common metadata is returned as a result set with one row for each column in the results metadata. Each row describes the type and nullability of the column in the format described in the following section. If the first statement does not exist for every control path, a result set with zero rows is returned.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_hidden	**bit**	Specifies whether the column is an extra column added for browsing information purposes that does not actually appear in the result set.
column_ordinal	**int**	Contains the ordinal position of the column in the result set. Position of the first column will be specified as 1.

COLUMN NAME	DATA TYPE	DESCRIPTION
name	**sysname**	Contains the name of the column if a name can be determined. Otherwise is NULL.
is_nullable	**bit**	Contains the value 1 if the column allows NULLs, 0 if the column does not allow NULLs, and 1 if it cannot be determined that the column allows NULLs.
system_type_id	**int**	Contains the system_type_id of the data type of the column as specified in sys.types. For CLR types, even though the system_type_name column will return NULL, this column will return the value 240.
system_type_name	**nvarchar(256)**	Contains the data type name. Includes arguments (such as length, precision, scale) specified for the data type of the column. If the data type is a user-defined alias type, the underlying system type is specified here. If it is a CLR user-defined type, NULL is returned in this column.
max_length	**smallint**	Maximum length (in bytes) of the column. -1 = Column data type is **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, or **xml**. For **text** columns, the **max_length** value will be 16 or the value set by **sp_tableoption 'text in row'**.
precision	**tinyint**	Precision of the column if numeric-based. Otherwise returns 0.
scale	**tinyint**	Scale of column if numeric-based. Otherwise returns 0.
collation_name	**sysname**	Name of the collation of the column if character-based. Otherwise returns NULL.
user_type_id	**int**	For CLR and alias types, contains the user_type_id of the data type of the column as specified in sys.types. Otherwise is NULL.
user_type_database	**sysname**	For CLR and alias types, contains the name of the database in which the type is defined. Otherwise is NULL.

COLUMN NAME	DATA TYPE	DESCRIPTION
user_type_schema	**sysname**	For CLR and alias types, contains the name of the schema in which the type is defined. Otherwise is NULL.
user_type_name	**sysname**	For CLR and alias types, contains the name of the type. Otherwise is NULL.
assembly_qualified_type_name	**nvarchar(4000)**	For CLR types, returns the name of the assembly and class defining the type. Otherwise is NULL.
xml_collection_id	**int**	Contains the xml_collection_id of the data type of the column as specified in sys.columns. This column will return NULL if the type returned is not associated with an XML schema collection.
xml_collection_database	**sysname**	Contains the database in which the XML schema collection associated with this type is defined. This column will return NULL if the type returned is not associated with an XML schema collection.
xml_collection_schema	**sysname**	Contains the schema in which the XML schema collection associated with this type is defined. This column will return NULL if the type returned is not associated with an XML schema collection.
xml_collection_name	**sysname**	Contains the name of the XML schema collection associated with this type. This column will return NULL if the type returned is not associated with an XML schema collection.
is_xml_document	**bit**	Returns 1 if the returned data type is XML and that type is guaranteed to be a complete XML document (including a root node), as opposed to an XML fragment). Otherwise returns 0.
is_case_sensitive	**bit**	Returns 1 if the column is of a case-sensitive string type and 0 if it is not.
is_fixed_length_clr_type	**bit**	Returns 1 if the column is of a fixed-length CLR type and 0 if it is not.

COLUMN NAME	DATA TYPE	DESCRIPTION
source_server	**sysname**	Name of the originating server returned by the column in this result (if it originates from a remote server). The name is given as it appears in sys.servers. Returns NULL if the column originates on the local server, or if it cannot be determined which server it originates on. Is only populated if browsing information is requested.
source_database	**sysname**	Name of the originating database returned by the column in this result. Returns NULL if the database cannot be determined. Is only populated if browsing information is requested.
source_schema	**sysname**	Name of the originating schema returned by the column in this result. Returns NULL if the schema cannot be determined. Is only populated if browsing information is requested.
source_table	**sysname**	Name of the originating table returned by the column in this result. Returns NULL if the table cannot be determined. Is only populated if browsing information is requested.
source_column	**sysname**	Name of the originating column returned by the column in this result. Returns NULL if the column cannot be determined. Is only populated if browsing information is requested.
is_identity_column	**bit**	Returns 1 if the column is an identity column and 0 if not. Returns NULL if it cannot be determined that the column is an identity column.
is_part_of_unique_key	**bit**	Returns 1 if the column is part of a unique index (including unique and primary constraint) and 0 if not. Returns NULL if it cannot be determined that the column is part of a unique index. Only populated if browsing information is requested.
is_updateable	**bit**	Returns 1 if the column is updateable and 0 if not. Returns NULL if it cannot be determined that the column is updateable.
is_computed_column	**bit**	Returns 1 if the column is a computed column and 0 if not. Returns NULL if it cannot be determined that the column is a computed column.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_sparse_column_set	**bit**	Returns 1 if the column is a sparse column and 0 if not. Returns NULL if it cannot be determined that the column is a part of a sparse column set.
ordinal_in_order_by_list	**smallint**	Position of this column in ORDER BY list Returns NULL if the column does not appear in the ORDER BY list or if the ORDER BY list cannot be uniquely determined.
order_by_list_length	**smallint**	Length of the ORDER BY list. Returns NULL if there is no ORDER BY list or if the ORDER BY list cannot be uniquely determined. Note that this value will be the same for all rows returned by sp_describe_first_result_set.
order_by_is_descending	**smallint NULL**	If the ordinal_in_order_by_list is not NULL, the **order_by_is_descending** column reports the direction of the ORDER BY clause for this column. Otherwise it reports NULL.
error_number	**int**	Contains the error number returned by the function. Contains NULL if no error occurred in the column.
error_severity	**int**	Contains the severity returned by the function. Contains NULL if no error occurred in the column.
error_state	**int**	Contains the state message returned by the function. If no error occurred. the column will contain NULL.
error_message	**nvarchar(4096)**	Contains the message returned by the function. If no error occurred, the column will contain NULL.
error_type	**int**	Contains an integer representing the error being returned. Maps to error_type_desc. See the list under remarks.
error_type_desc	**nvarchar(60)**	Contains a short uppercase string representing the error being returned. Maps to error_type. See the list under remarks.

Remarks

This function uses the same algorithm as **sp_describe_first_result_set**. For more information, see sp_describe_first_result_set (Transact-SQL).

The following table lists the error types and their descriptions

ERROR_TYPE	ERROR_TYPE	DESCRIPTION
1	MISC	All errors that are not otherwise described.
2	SYNTAX	A syntax error occurred in the batch.
3	CONFLICTING_RESULTS	The result could not be determined because of a conflict between two possible first statements.
4	DYNAMIC_SQL	The result could not be determined because of dynamic SQL that could potentially return the first result.
5	CLR_PROCEDURE	The result could not be determined because a CLR stored procedure could potentially return the first result.
6	CLR_TRIGGER	The result could not be determined because a CLR trigger could potentially return the first result.
7	EXTENDED_PROCEDURE	The result could not be determined because an extended stored procedure could potentially return the first result.
8	UNDECLARED_PARAMETER	The result could not be determined because the data type of one or more of the result set's columns potentially depends on an undeclared parameter.
9	RECURSION	The result could not be determined because the batch contains a recursive statement.
10	TEMPORARY_TABLE	The result could not be determined because the batch contains a temporary table and is not supported by **sp_describe_first_result_set** .
11	UNSUPPORTED_STATEMENT	The result could not be determined because the batch contains a statement that is not supported by **sp_describe_first_result_set** (e.g., FETCH, REVERT etc.).
12	OBJECT_ID_NOT_SUPPORTED	The @object_id passed to the function is not supported (i.e. not a stored procedure)
13	OBJECT_ID_DOES_NOT_EXIST	The @object_id passed to the function was not found in the system catalog.

Permissions

Requires permission to execute the @tsql argument.

Examples

A. Returning metadata with and without browse information

The following example creates a stored procedure named TestProc2 that returns two result sets. Then the example demonstrates that **sys.dm_exec_describe_first_result_set** returns information about the first result set in the procedure, with and without the browse information.

```
CREATE PROC TestProc2
AS
SELECT object_id, name FROM sys.objects ;
SELECT name, schema_id, create_date FROM sys.objects ;
GO

SELECT * FROM sys.dm_exec_describe_first_result_set_for_object(OBJECT_ID('TestProc2'), 0) ;
SELECT * FROM sys.dm_exec_describe_first_result_set_for_object(OBJECT_ID('TestProc2'), 1) ;
GO
```

B. Combining the sys.dm_exec_describe_first_result_set_for_object function and a table or view

The following example uses both the sys.procedures system catalog view and the **sys.dm_exec_describe_first_result_set_for_object** function to display metadata for the result sets of all stored procedures in the **AdventureWorks2012** database.

```
USE AdventureWorks2012;
GO

SELECT p.name, r.*
FROM sys.procedures AS p
CROSS APPLY sys.dm_exec_describe_first_result_set_for_object(p.object_id, 0) AS r;
GO
```

See Also

sp_describe_first_result_set (Transact-SQL)
sp_describe_undeclared_parameters (Transact-SQL)
sys.dm_exec_describe_first_result_set (Transact-SQL)

sys.dm_exec_distributed_request_steps (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all steps that compose a given PolyBase request or query. It lists one row per query step.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
execution_id	**int**	execution_id and step_index make up the key for this view. Unique numeric id associated with the request.	See ID in sys.dm_exec_requests (Transact-SQL).
step_index	**int**	The position of this step in the sequence of steps that make up the request.	0 to (n-1) for a request with n steps.
operation_type	**nvarchar(128)**	Type of the operation represented by this step.	'MoveOperation','OnOperation','RandomIDOperation','RemoteOperation','ReturnOperation','ShuffleMoveOperation','TempTablePropertiesOperation','DropDiagnosticsNotifyOperation', 'HadoopShuffleOperation', 'HadoopBroadCastOperation', 'HadoopRoundRobinOperation'
distribution_type	**nvarchar(32)**	Where the step is executing.	'AllComputeNodes','AllDistributions','ComputeNode','Distribution','AllNodes','SubsetNodes','SubsetDistributions','Unspecified'.
location_type	**nvarchar(32)**	Where the step is executing.	'Compute','Head' or 'DMS'. All data movement steps show 'DMS'.
status	**nvarchar(32)**	Status of this step	'Pending', 'Running', 'Complete', 'Failed', 'UndoFailed', 'PendingCancel', 'Cancelled', 'Undone', 'Aborted'
error_id	**nvarchar(36)**	Unique id of the error associated with this step, if any	See id of sys.dm_exec_compute_node_errors (Transact-SQL), NULL if no error occurred.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
start_time	**datetime**	Time at which the step started execution	Smaller or equal to current time and larger or equal to end_compile_time of the query to which this step belongs.
end_time	**datetime**	Time at which this step completed execution, was cancelled, or failed.	Smaller or equal to current time and larger or equal to start_time, set to NULL for steps currently in execution or queued.
total_elapsed_time	**int**	Total amount of time the query step has been executing, in milliseconds	Between 0 and the difference between end_time and start_time. 0 for queued steps.
row_count	**bigint**	Total number of rows changed or returned by this request	0 for steps that did not change or return data, number of rows affected otherwise. Set to -1 for DMS steps.
command	nvarchar(4000)	Holds the full text of the command of this step.	Any valid request string for a step. Truncated if longer than 4000 characters.

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_distributed_requests (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all requests currently or recently active in PolyBase queries. It lists one row per request/query.

Based on session and request ID, a user can then retrieve the actual distributed requests generated to be executed – via sys.dm_exec_distributed_requests. For example, a query involving regular SQL and external SQL tables will be decomposed into various statements/requests executed across the various compute nodes. To track the distributed steps across all compute nodes, we introduce a 'global' execution ID which can be used to track all operations on the compute nodes associated with one particular request and operator, respectively.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
sql_handle	**varbinary(64)**	Key for this view. Unique numeric id associated with the request.	Unique across all requests in the system.
execution_id	**nvarchar(32**	Unique numeric id associated with the session in which this query was run.	
status	**nvarchar(32**	Current status of the request.	'Pending', 'Authorizing', 'AcquireSystemResources', 'Initializing', 'Plan', 'Parsing', 'AquireResources', 'Running', 'Cancelling', 'Complete', 'Failed', 'Cancelled'.
error_id	**nvarchar(36)**	Unique id of the error associated with the request, if any.	Set to NULL if no error occurred.
start_time	**datetime**	Time at which the request execution was started.	0 for queued requests; otherwise, valid datetime smaller or equal to current time.
end_time	**datetime**	Time at which the engine completed compiling the request.	Null for queued or active requests; otherwise, a valid datetime smaller or equal to current time.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
total_elapsed_time	**int**	Time elapsed in execution since the request was started, in milliseconds.	Between 0 and the difference between start_time and end_time.If total_elapsed_time exceeds the maximum value for an integer, total_elapsed_time will continue to be the maximum value. This condition will generate the warning "The maximum value has been exceeded." The maximum value in milliseconds is equivalent to 24.8 days.

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_distributed_sql_requests (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all SQL query distributions as part of a SQL step in the query. This view shows the data for the last 1000 requests; active requests always have the data present in this view.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
execution_id	**nvarchar(32)**	execution_id and step_index make up the key for this view. Unique numeric id associated with the request.	See ID in sys.dm_exec_requests (Transact-SQL)
step_index	**int**	Index of the query step this distribution is part of.	See step_index in sys.dm_exec_distributed_request_steps (Transact-SQL).
compute_node_id	**int**	Type of the operation represented by this step.	See compute_node_id in sys.dm_exec_compute_nodes (Transact-SQL).
distribution_id	**int**	Where the step is executing.	Set to -1 for requests that run at the node scope not the distribution scope.
status	**nvarchar(32)**	Status of this step	Active, Cancelled, Completed, Failed, Queued
error_id	**nvarchar(36)**	Unique id of the error associated with this step, if any	See id of sys.dm_exec_compute_node_errors (Transact-SQL), NULL if no error occurred.
start_time	**datetime**	Time at which the step started execution	Smaller or equal to current time and larger or equal to end_compile_time of the query to which this step belongs.
end_time	**datetime**	Time at which this step completed execution, was cancelled, or failed.	Smaller or equal to current time and larger or equal to start_time, set to NULL for steps currently in execution or queued.
total_elapsed_time	**int**	Total amount of time the query step has been executing, in milliseconds	Between 0 and the difference between end_time and start_time. 0 for queued steps.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
row_count	**bigint**	Total number of rows changed or returned by this request	0 for steps that did not change or return data, number of rows affected otherwise. Set to -1 for DMS steps.
spid	**int**	Session id on the SQL Server instance executing the query distribution	
command	nvarchar(4000)	Holds the full text of the command of this step.	Any valid request string for a step. Truncated if longer than 4000 characters.

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_dms_services (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all of the DMS services running on the PolyBase compute nodes. It lists one row per service instance.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
dms_core_id	**int**	Unique numeric id associated with the DMS core. Key for this view.	Unique ID.
compute_node_id	**int**	ID of the node on which this DMS service is running	See *compute_node_id* in sys.dm_exec_compute_nodes (Transact-SQL).
status	**nvarchar(32)**	Current status of the DMS service	

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_dms_workers (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all workers completing DMS steps.

This view shows the data for the last 1000 requests and active requests; active requests always have the data present in this view.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
execution_id	**nvarchar(32)**	Query that this DMS worker is part of.request_id, step_index, and dms_step_index form the key for this view.	
step_index	**int**	Query step this DMS worker is part of.	See step index in sys.dm_exec_distributed_request_steps (Transact-SQL).
dms_step_index	**int**	Step in the DMS plan that this worker is running.	See sys.dm_exec_dms_workers (Transact-SQL)
compute_node_id	**int**	Node that the worker is running on.	See sys.dm_exec_compute_nodes (Transact-SQL).
distribution_id	**int**		
type	**nvarcha(32)**		
status	**nvarchar(32)**	Status of this step	'Pending', 'Running', 'Complete', 'Failed', 'UndoFailed', 'PendingCancel', 'Cancelled', 'Undone', 'Aborted'
bytes_per_sec	**bigint**		
bytes_processed	**bigint**		
rows_processed	**bigint**		
start_time	**datetime**	Time at which the step started execution	Smaller or equal to current time and larger or equal to end_compile_time of the query to which this step belongs.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
end_time	**datetime**	Time at which this step completed execution, was cancelled, or failed.	Smaller or equal to current time and larger or equal to start_time, set to NULL for steps currently in execution or queued.
total_elapsed_time	**int**	Total amount of time the query step has been executing, in milliseconds	Between 0 and the difference between end_time and start_time. 0 for queued steps.
cpu_time	**bigint**		
query_time	**int**		
buffers_available	**int**		
dms_cpid	**int**		
sql_spid	**int**		
error_id	**nvarchar(36)**		
source_info	**nvarchar(4000)**		
destination_info	**nvarchar(4000)**		
command	**nvarchar(4000)**		

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_external_operations (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Captures information about external PolyBase operations.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
execution_id	**nvarchar(32)**	Unique query identifier associated with PolyBase query	See ID in sys.dm_exec_requests (Transact-SQL)
step_index	**int**	Index of the query step	See step_index in sys.dm_exec_distributed_request_steps (Transact-SQL)
operation_ type	**nvarchar(128)**	Describes a Hadoop operation or other external operation	'External Hadoop Operation'
operation_ name	**nvarchar(4000)**	Indicates how the status of job in percentage (how much is the input consumed)	0-1 – multiplied by factor 100 (completed)
map_ progress	**float**	Indicates how the status of a reduce job in percentage, if any	0-1 – multiplied by factor 100 (completed)

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_external_work (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the workload per worker, on each compute node.

Query sys.dm_exec_external_work to identify the work spun up to communicate with the external data source (e.g. Hadoop or external SQL Server).

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
execution_id	**nvarchar(32)**	Unique identifier for associated PolyBase query.	See *request_ID* in sys.dm_exec_requests (Transact-SQL).
step_index	**int**	The request this worker is performing.	See *step_index* in sys.dm_exec_requests (Transact-SQL).
dms_step_index	**int**	Step in the DMS plan that this worker is executing.	See sys.dm_exec_dms_workers (Transact-SQL).
compute_node_id	**int**	The node the worker is running on.	See sys.dm_exec_compute_nodes (Transact-SQL).
type	**nvarchar(60)**	The type of external work.	'File Split'
work_id	**int**	ID of the actual split.	Greater than or equal to 0.
input_name	**nvarchar(4000)**	Name of the input to be read	File name when using Hadoop.
read_location	**bigint**	Offset or read location.	Offset of the file to read.
bytes_processed	**bigint**	Total bytes processed by this worker.	Greater than or equal to 0.
length	**bigint**	Length of the split or HDFS block in case of Hadoop	User-definable. The default is 64M
status	**nvarchar(32)**	Status of the worker	Pending, Processing, Done, Failed, Aborted
start_time	**datetime**	Beginning of the work	
end_time	**datetime**	End of the work	
total_elapsed_time	**int**	Total time in milliseconds	

See Also

PolyBase troubleshooting with dynamic management views
Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_function_stats (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns aggregate performance statistics for cached functions. The view returns one row for each cached function plan, and the lifetime of the row is as long as the function remains cached. When a function is removed from the cache, the corresponding row is eliminated from this view. At that time, a Performance Statistics SQL trace event is raised similar to **sys.dm_exec_query_stats**. Returns information about scalar functions, including in-memory functions and CLR scalar functions. Does not return information about table valued functions.

In Azure SQL Database, dynamic management views cannot expose information that would impact database containment or expose information about other databases the user has access to. To avoid exposing this information, every row that contains data that doesn't belong to the connected tenant is filtered out.

> **NOTE**
> An initial query of **sys.dm_exec_function_stats** might produce inaccurate results if there is a workload currently executing on the server. More accurate results may be determined by rerunning the query.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	Database ID in which the function resides.
object_id	**int**	Object identification number of the function.
type	**char(2)**	Type of the object: FN = Scalar valued functions
type_desc	**nvarchar(60)**	Description of the object type: SQL_SCALAR_FUNCTION
sql_handle	**varbinary(64)**	This can be used to correlate with queries in **sys.dm_exec_query_stats** that were executed from within this function.
plan_handle	**varbinary(64)**	Identifier for the in-memory plan. This identifier is transient and remains constant only while the plan remains in the cache. This value may be used with the **sys.dm_exec_cached_plans** dynamic management view. Will always be 0x000 when a natively compiled function queries a memory-optimized table.

COLUMN NAME	DATA TYPE	DESCRIPTION
cached_time	**datetime**	Time at which the function was added to the cache.
last_execution_time	**datetime**	Last time at which the function was executed.
execution_count	**bigint**	Number of times that the function has been executed since it was last compiled.
total_worker_time	**bigint**	Total amount of CPU time, in microseconds, that was consumed by executions of this function since it was compiled. For natively compiled functions, **total_worker_time** may not be accurate if many executions take less than 1 millisecond.
last_worker_time	**bigint**	CPU time, in microseconds, that was consumed the last time the function was executed. [1]
min_worker_time	**bigint**	Minimum CPU time, in microseconds, that this function has ever consumed during a single execution. [1]
max_worker_time	**bigint**	Maximum CPU time, in microseconds, that this function has ever consumed during a single execution. [1]
total_physical_reads	**bigint**	Total number of physical reads performed by executions of this function since it was compiled. Will always be 0 querying a memory-optimized table.
last_physical_reads	**bigint**	Number of physical reads performed the last time the function was executed. Will always be 0 querying a memory-optimized table.
min_physical_reads	**bigint**	Minimum number of physical reads that this function has ever performed during a single execution. Will always be 0 querying a memory-optimized table.

COLUMN NAME	DATA TYPE	DESCRIPTION
max_physical_reads	**bigint**	Maximum number of physical reads that this function has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
total_logical_writes	**bigint**	Total number of logical writes performed by executions of this function since it was compiled. Will always be 0 querying a memory-optimized table.
last_logical_writes	**bigint**	Number of the number of buffer pool pages dirtied the last time the plan was executed. If a page is already dirty (modified) no writes are counted. Will always be 0 querying a memory-optimized table.
min_logical_writes	**bigint**	Minimum number of logical writes that this function has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
max_logical_writes	**bigint**	Maximum number of logical writes that this function has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
total_logical_reads	**bigint**	Total number of logical reads performed by executions of this function since it was compiled. Will always be 0 querying a memory-optimized table.
last_logical_reads	**bigint**	Number of logical reads performed the last time the function was executed. Will always be 0 querying a memory-optimized table.
min_logical_reads	**bigint**	Minimum number of logical reads that this function has ever performed during a single execution. Will always be 0 querying a memory-optimized table.

COLUMN NAME	DATA TYPE	DESCRIPTION
max_logical_reads	**bigint**	Maximum number of logical reads that this function has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
total_elapsed_time	**bigint**	Total elapsed time, in microseconds, for completed executions of this function.
last_elapsed_time	**bigint**	Elapsed time, in microseconds, for the most recently completed execution of this function.
min_elapsed_time	**bigint**	Minimum elapsed time, in microseconds, for any completed execution of this function.
max_elapsed_time	**bigint**	Maximum elapsed time, in microseconds, for any completed execution of this function.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example returns information about the top ten functions identified by average elapsed time.

```
SELECT TOP 10 d.object_id, d.database_id, OBJECT_NAME(object_id, database_id) 'function name',
    d.cached_time, d.last_execution_time, d.total_elapsed_time,
    d.total_elapsed_time/d.execution_count AS [avg_elapsed_time],
    d.last_elapsed_time, d.execution_count
FROM sys.dm_exec_function_stats AS d
ORDER BY [total_worker_time] DESC;
```

See Also

Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_sql_text (Transact-SQL)
sys.dm_exec_query_stats (Transact-SQL)

sys.dm_exec_trigger_stats (Transact-SQL)
sys.dm_exec_procedure_stats (Transact-SQL)

sys.dm_exec_input_buffer (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014 SP2) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about statements submitted to an instance of SQL Server.

Syntax

```
sys.dm_exec_input_buffer ( session_id , request_id )
```

Arguments

session_id
Is the session id executing the batch to be looked up. *session_id* is **smallint**. *session_id* can be obtained from the following dynamic management objects:

- sys.dm_exec_requests
- sys.dm_exec_sessions
- sys.dm_exec_connections

request_id
The request_id from sys.dm_exec_requests. *request_id* is **int**.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
event_type	**nvarchar(256)**	The type of event in the input buffer for the given spid.
parameters	**smallint**	Any parameters provided for the statement.
event_info	**nvarchar(max)**	The text of the statement in the input buffer for the given spid.

Permissions

On SQL Server, if the user has VIEW SERVER STATE permission, the user will see all executing sessions on the instance of SQL Server; otherwise, the user will see only the current session.

On SQL Database, if the user is the database owner, the user will see all executing sessions on the SQL Database; otherwise, the user will see only the current session.

Remarks

This dynamic management function can be used in conjunction with sys.dm_exec_sessions or sys.dm_exec_requests by doing **CROSS APPLY**.

Examples

A. Simple example

The following example demonstrates passing a session id (SPID) and a request id to the function.

```
SELECT * FROM sys.dm_exec_input_buffer (52, 0);
GO
```

B. Using cross apply to additional information

The following example lists the input buffer for sessions with session id greater than 50.

```
SELECT es.session_id, ib.event_info
FROM sys.dm_exec_sessions AS es
CROSS APPLY sys.dm_exec_input_buffer(es.session_id, NULL) AS ib
WHERE es.session_id > 50;
GO
```

See Also

Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_sessions (Transact-SQL)
sys.dm_exec_requests (Transact-SQL)
DBCC INPUTBUFFER (Transact-SQL)

sys.dm_exec_plan_attributes (Transact-SQL)

5/4/2018 • 7 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row per plan attribute for the plan specified by the plan handle. You can use this table-valued function to get details about a particular plan, such as the cache key values or the number of current simultaneous executions of the plan.

> **NOTE**
>
> Some of the information returned through this function maps to the sys.syscacheobjects backward compatibility view.

Syntax

```
sys.dm_exec_plan_attributes ( plan_handle )
```

Arguments

plan_handle
Uniquely identifies a query plan for a batch that has executed and whose plan resides in the plan cache. *plan_handle* is **varbinary(64)**. The plan handle can be obtained from the sys.dm_exec_cached_plans dynamic management view.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
attribute	**varchar(128)**	Name of the attribute associated with this plan. The table immediately below this one lists the possible attributes, their data types, and their descriptions.
value	**sql_variant**	Value of the attribute that is associated with this plan.
is_cache_key	**bit**	Indicates whether the attribute is used as part of the cache lookup key for the plan.

From the above table, **attribute** can have the following values:

ATTRIBUTE	DATA TYPE	DESCRIPTION
set_options	**int**	Indicates the option values that the plan was compiled with.

ATTRIBUTE	DATA TYPE	DESCRIPTION
objectid	**int**	One of the main keys used for looking up an object in the cache. This is the object ID stored in sys.objects for database objects (procedures, views, triggers, and so on). For plans of type "Adhoc" or "Prepared", it is an internal hash of the batch text.
dbid	**int**	Is the ID of the database containing the entity the plan refers to. For ad hoc or prepared plans, it is the database ID from which the batch is executed.
dbid_execute	**int**	For system objects stored in the **Resource** database, the database ID from which the cached plan is executed. For all other cases, it is 0.
user_id	**int**	Value of -2 indicates that the batch submitted does not depend on implicit name resolution and can be shared among different users. This is the preferred method. Any other value represents the user ID of the user submitting the query in the database.
language_id	**smallint**	ID of the language of the connection that created the cache object. For more information, see sys.syslanguages (Transact-SQL).
date_format	**smallint**	Date format of the connection that created the cache object. For more information, see SET DATEFORMAT (Transact-SQL).
date_first	**tinyint**	Date first value. For more information, see SET DATEFIRST (Transact-SQL).
status	**int**	Internal status bits that are part of the cache lookup key.
required_cursor_options	**int**	Cursor options specified by the user such as the cursor type.
acceptable_cursor_options	**int**	Cursor options that SQL Server may implicitly convert to in order to support the execution of the statement. For example, the user may specify a dynamic cursor, but the query optimizer is permitted to convert this cursor type to a static cursor.
inuse_exec_context	**int**	Number of currently executing batches that are using the query plan.

ATTRIBUTE	DATA TYPE	DESCRIPTION
free_exec_context	**int**	Number of cached execution contexts for the query plan that are not being currently used.
hits_exec_context	**int**	Number of times the execution context was obtained from the plan cache and reused, saving the overhead of recompiling the SQL statement. The value is an aggregate for all batch executions so far.
misses_exec_context	**int**	Number of times that an execution context could not be found in the plan cache, resulting in the creation of a new execution context for the batch execution.
removed_exec_context	**int**	Number of execution contexts that have been removed because of memory pressure on the cached plan.
inuse_cursors	**int**	Number of currently executing batches containing one or more cursors that are using the cached plan.
free_cursors	**int**	Number of idle or free cursors for the cached plan.
hits_cursors	**int**	Number of times that an inactive cursor was obtained from the cached plan and reused. The value is an aggregate for all batch executions so far.
misses_cursors	**int**	Number of times that an inactive cursor could not be found in the cache.
removed_cursors	**int**	Number of cursors that have been removed because of memory pressure on the cached plan.
sql_handle	**varbinary**(64)	The SQL handle for the batch.

ATTRIBUTE	DATA TYPE	DESCRIPTION
merge_action_type	**smallint**	The type of trigger execution plan used as the result of a MERGE statement. 0 indicates a non-trigger plan, a trigger plan that does not execute as the result of a MERGE statement, or a trigger plan that executes as the result of a MERGE statement that only specifies a DELETE action. 1 indicates an INSERT trigger plan that runs as the result of a MERGE statement. 2 indicates an UPDATE trigger plan that runs as the result of a MERGE statement. 3 indicates a DELETE trigger plan that runs as the result of a MERGE statement containing a corresponding INSERT or UPDATE action. For nested triggers run by cascading actions, this value is the action of the MERGE statement that caused the cascade.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

Set Options

Copies of the same compiled plan might differ only by the value in the **set_options** column. This indicates that different connections are using different sets of SET options for the same query. Using different sets of options is usually undesirable because it can cause extra compilations, less plan reuse, and plan cache inflation because of multiple copies of plans in the cache.

Evaluating Set Options

To translate the value returned in **set_options** to the options with which the plan was compiled, subtract the values from the **set_options** value, starting with the largest possible value, until you reach 0. Each value you subtract corresponds to an option that was used in the query plan. For example, if the value in **set_options** is 251, the options the plan was compiled with are ANSI_NULL_DFLT_ON (128), QUOTED_IDENTIFIER (64), ANSI_NULLS(32), ANSI_WARNINGS (16), CONCAT_NULL_YIELDS_NULL (8), Parallel Plan(2) and ANSI_PADDING (1).

OPTION	VALUE
ANSI_PADDING	1

OPTION	VALUE
Parallel Plan	2
FORCEPLAN	4
CONCAT_NULL_YIELDS_NULL	8
ANSI_WARNINGS	16
ANSI_NULLS	32
QUOTED_IDENTIFIER	64
ANSI_NULL_DFLT_ON	128
ANSI_NULL_DFLT_OFF	256
NoBrowseTable Indicates that the plan does not use a work table to implement a FOR BROWSE operation.	512
TriggerOneRow Indicates that the plan contains single row optimization for AFTER trigger delta tables.	1024
ResyncQuery Indicates that the query was submitted by internal system stored procedures.	2048
ARITH_ABORT	4096
NUMERIC_ROUNDABORT	8192
DATEFIRST	16384
DATEFORMAT	32768
LanguageID	65536
UPON Indicates that the database option PARAMETERIZATION was set to FORCED when the plan was compiled.	131072
ROWCOUNT	**Applies To:** SQL Server 2012 (11.x) to SQL Server 2017 262144

Cursors

Inactive cursors are cached in a compiled plan so that the memory used to store the cursor can be reused by concurrent users of cursors. For example, suppose that a batch declares and uses a cursor without deallocating it. If there are two users executing the same batch, there will be two active cursors. Once the cursors are deallocated (potentially in different batches), the memory used to store the cursor is cached and not released. This list of inactive cursors is kept in the compiled plan. The next time a user executes the batch, the cached cursor memory will be reused and initialized appropriately as an active cursor.

Evaluating Cursor Options

To translate the value returned in **required_cursor_options** and **acceptable_cursor_options** to the options with which the plan was compiled, subtract the values from the column value, starting with the largest possible value, until you reach 0. Each value you subtract corresponds to a cursor option that was used in the query plan.

OPTION	VALUE
None	0
INSENSITIVE	1
SCROLL	2
READ ONLY	4
FOR UPDATE	8
LOCAL	16
GLOBAL	32
FORWARD_ONLY	64
KEYSET	128
DYNAMIC	256
SCROLL_LOCKS	512
OPTIMISTIC	1024
STATIC	2048
FAST_FORWARD	4096
IN PLACE	8192
FOR *select_statement*	16384

Examples

A. Returning the attributes for a specific plan

The following example returns all plan attributes for a specified plan. The `sys.dm_exec_cached_plans` dynamic management view is queried first to obtain the plan handle for the specified plan. In the second query, replace `<plan_handle>` with a plan handle value from the first query.

```
SELECT plan_handle, refcounts, usecounts, size_in_bytes, cacheobjtype, objtype
FROM sys.dm_exec_cached_plans;
GO
SELECT attribute, value, is_cache_key
FROM sys.dm_exec_plan_attributes(<plan_handle>);
GO
```

B. Returning the SET options for compiled plans and the SQL handle for cached plans

The following example returns a value representing the options that each plan was compiled with. In addition, the SQL handle for all the cached plans is returned.

```
SELECT plan_handle, pvt.set_options, pvt.sql_handle
FROM (
    SELECT plan_handle, epa.attribute, epa.value
    FROM sys.dm_exec_cached_plans
        OUTER APPLY sys.dm_exec_plan_attributes(plan_handle) AS epa
    WHERE cacheobjtype = 'Compiled Plan') AS ecpa
PIVOT (MAX(ecpa.value) FOR ecpa.attribute IN ("set_options", "sql_handle")) AS pvt;
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_cached_plans (Transact-SQL)
sys.databases (Transact-SQL)
sys.objects (Transact-SQL)

sys.dm_exec_procedure_stats (Transact-SQL)

5/4/2018 • 5 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns aggregate performance statistics for cached stored procedures. The view returns one row for each cached stored procedure plan, and the lifetime of the row is as long as the stored procedure remains cached. When a stored procedure is removed from the cache, the corresponding row is eliminated from this view. At that time, a Performance Statistics SQL trace event is raised similar to **sys.dm_exec_query_stats**.

In Azure SQL Database, dynamic management views cannot expose information that would impact database containment or expose information about other databases the user has access to. To avoid exposing this information, every row that contains data that doesn't belong to the connected tenant is filtered out.

> **NOTE**
>
> An initial query of **sys.dm_exec_procedure_stats** might produce inaccurate results if there is a workload currently executing on the server. More accurate results may be determined by rerunning the query.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_exec_procedure_stats**.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	Database ID in which the stored procedure resides.
object_id	**int**	Object identification number of the stored procedure.
type	**char(2)**	Type of the object: P = SQL stored procedure PC = Assembly (CLR) stored procedure X = Extended stored procedure
type_desc	**nvarchar(60)**	Description of the object type: SQL_STORED_PROCEDURE CLR_STORED_PROCEDURE EXTENDED_STORED_PROCEDURE

COLUMN NAME	DATA TYPE	DESCRIPTION
sql_handle	**varbinary(64)**	This can be used to correlate with queries in **sys.dm_exec_query_stats** that were executed from within this stored procedure.
plan_handle	**varbinary(64)**	Identifier for the in-memory plan. This identifier is transient and remains constant only while the plan remains in the cache. This value may be used with the **sys.dm_exec_cached_plans** dynamic management view. Will always be 0x000 when a natively compiled stored procedure queries a memory-optimized table.
cached_time	**datetime**	Time at which the stored procedure was added to the cache.
last_execution_time	**datetime**	Last time at which the stored procedure was executed.
execution_count	**bigint**	The number of times that the stored procedure has been executed since it was last compiled.
total_worker_time	**bigint**	The total amount of CPU time, in microseconds, that was consumed by executions of this stored procedure since it was compiled. For natively compiled stored procedures, **total_worker_time** may not be accurate if many executions take less than 1 millisecond.
last_worker_time	**bigint**	CPU time, in microseconds, that was consumed the last time the stored procedure was executed. [1]
min_worker_time	**bigint**	The minimum CPU time, in microseconds, that this stored procedure has ever consumed during a single execution. [1]
max_worker_time	**bigint**	The maximum CPU time, in microseconds, that this stored procedure has ever consumed during a single execution. [1]
total_physical_reads	**bigint**	The total number of physical reads performed by executions of this stored procedure since it was compiled. Will always be 0 querying a memory-optimized table.

COLUMN NAME	DATA TYPE	DESCRIPTION
last_physical_reads	**bigint**	The number of physical reads performed the last time the stored procedure was executed. Will always be 0 querying a memory-optimized table.
min_physical_reads	**bigint**	The minimum number of physical reads that this stored procedure has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
max_physical_reads	**bigint**	The maximum number of physical reads that this stored procedure has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
total_logical_writes	**bigint**	The total number of logical writes performed by executions of this stored procedure since it was compiled. Will always be 0 querying a memory-optimized table.
last_logical_writes	**bigint**	The number of buffer pool pages dirtied the last time the plan was executed. If a page is already dirty (modified) no writes are counted. Will always be 0 querying a memory-optimized table.
min_logical_writes	**bigint**	The minimum number of logical writes that this stored procedure has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
max_logical_writes	**bigint**	The maximum number of logical writes that this stored procedure has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
total_logical_reads	**bigint**	The total number of logical reads performed by executions of this stored procedure since it was compiled. Will always be 0 querying a memory-optimized table.

COLUMN NAME	DATA TYPE	DESCRIPTION
last_logical_reads	**bigint**	The number of logical reads performed the last time the stored procedure was executed. Will always be 0 querying a memory-optimized table.
min_logical_reads	**bigint**	The minimum number of logical reads that this stored procedure has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
max_logical_reads	**bigint**	The maximum number of logical reads that this stored procedure has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
total_elapsed_time	**bigint**	The total elapsed time, in microseconds, for completed executions of this stored procedure.
last_elapsed_time	**bigint**	The elapsed time, in microseconds, for the most recently completed execution of this stored procedure.
min_elapsed_time	**bigint**	The minimum elapsed time, in microseconds, for any completed execution of this stored procedure.
max_elapsed_time	**bigint**	The maximum elapsed time, in microseconds, for any completed execution of this stored procedure.
total_spills	**bigint**	The total number of pages spilled by execution of this stored procedure since it was compiled. **Applies to**: Starting with SQL Server 2017 (14.x) CU3
last_spills	**bigint**	The number of pages spilled the last time the stored procedure was executed. **Applies to**: Starting with SQL Server 2017 (14.x) CU3
min_spills	**bigint**	The minimum number of pages that this stored procedure has ever spilled during a single execution. **Applies to**: Starting with SQL Server 2017 (14.x) CU3

COLUMN NAME	DATA TYPE	DESCRIPTION
max_spills	**bigint**	The maximum number of pages that this stored procedure has ever spilled during a single execution. **Applies to**: Starting with SQL Server 2017 (14.x) CU3
pdw_node_id	**int**	The identifier for the node that this distribution is on. **Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse

[1] For natively compiled stored procedures when statistics collection is enabled, worker time is collected in milliseconds. If the query executes in less than a millisecond, the value will be 0.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

Statistics in the view are updated when a stored procedure execution completes.

Examples

The following example returns information about the top ten stored procedures identified by average elapsed time.

```
SELECT TOP 10 d.object_id, d.database_id, OBJECT_NAME(object_id, database_id) 'proc name',
    d.cached_time, d.last_execution_time, d.total_elapsed_time,
    d.total_elapsed_time/d.execution_count AS [avg_elapsed_time],
    d.last_elapsed_time, d.execution_count
FROM sys.dm_exec_procedure_stats AS d
ORDER BY [total_worker_time] DESC;
```

See Also

Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_sql_text (Transact-SQL)
sys.dm_exec_query_plan (Transact-SQL)
sys.dm_exec_query_stats (Transact-SQL)
sys.dm_exec_trigger_stats (Transact-SQL)
sys.dm_exec_cached_plans (Transact-SQL)

sys.dm_exec_query_memory_grants (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about all queries that have requested and are waiting for a memory grant or have been given a memory grant. Queries that do not require a memory grant will not appear in this view. For example, sort and hash join operations have memory grants for query execution, while queries without an **ORDER BY** clause will not have a memory grant.

In Azure SQL Database, dynamic management views cannot expose information that would impact database containment or expose information about other databases the user has access to. To avoid exposing this information, every row that contains data that doesn't belong to the connected tenant is filtered out. In addition, the values in the columns **scheduler_id**, **wait_order**, **pool_id**, **group_id** are filtered; the column value is set to NULL.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_exec_query_memory_grants**.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**smallint**	ID (SPID) of the session where this query is running.
request_id	**int**	ID of the request. Unique in the context of the session.
scheduler_id	**int**	ID of the scheduler that is scheduling this query.
dop	**smallint**	Degree of parallelism of this query.
request_time	**datetime**	Date and time when this query requested the memory grant.
grant_time	**datetime**	Date and time when memory was granted for this query. NULL if memory is not granted yet.
requested_memory_kb	**bigint**	Total requested amount of memory in kilobytes.

COLUMN NAME	DATA TYPE	DESCRIPTION
granted_memory_kb	**bigint**	Total amount of memory actually granted in kilobytes. Can be NULL if the memory is not granted yet. For a typical situation, this value should be the same as **requested_memory_kb**. For index creation, the server may allow additional on-demand memory beyond initially granted memory.
required_memory_kb	**bigint**	Minimum memory required to run this query in kilobytes. **requested_memory_kb** is the same or larger than this amount.
used_memory_kb	**bigint**	Physical memory used at this moment in kilobytes.
max_used_memory_kb	**bigint**	Maximum physical memory used up to this moment in kilobytes.
query_cost	**float**	Estimated query cost.
timeout_sec	**int**	Time-out in seconds before this query gives up the memory grant request.
resource_semaphore_id	**smallint**	Non-unique ID of the resource semaphore on which this query is waiting. **Note:** This ID is unique in versions of SQL Server that are earlier than SQL Server 2008. This change can affect troubleshooting query execution. For more information, see the "Remarks" section later in this topic.
queue_id	**smallint**	ID of waiting queue where this query waits for memory grants. NULL if the memory is already granted.
wait_order	**int**	Sequential order of waiting queries within the specified **queue_id**. This value can change for a given query if other queries get memory grants or time out. NULL if memory is already granted.
is_next_candidate	**bit**	Candidate for next memory grant. 1 = Yes 0 = No NULL = Memory is already granted.
wait_time_ms	**bigint**	Wait time in milliseconds. NULL if the memory is already granted.

COLUMN NAME	DATA TYPE	DESCRIPTION
plan_handle	**varbinary(64)**	Identifier for this query plan. Use **sys.dm_exec_query_plan** to extract the actual XML plan.
sql_handle	**varbinary(64)**	Identifier for Transact-SQL text for this query. Use **sys.dm_exec_sql_text** to get the actual Transact-SQL text.
group_id	**int**	ID for the workload group where this query is running.
pool_id	**int**	ID of the resource pool that this workload group belongs to.
is_small	**tinyint**	When set to 1, indicates that this grant uses the small resource semaphore. When set to 0, indicates that a regular semaphore is used.
ideal_memory_kb	**bigint**	Size, in kilobytes (KB), of the memory grant to fit everything into physical memory. This is based on the cardinality estimate.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

A typical debugging scenario for query time-out may look like the following:

- Check overall system memory status using sys.dm_os_memory_clerks, sys.dm_os_sys_info, and various performance counters.
- Check for query-execution memory reservations in **sys.dm_os_memory_clerks** where `type = 'MEMORYCLERK_SQLQERESERVATIONS'`.
- Check for queries waiting for grants using **sys.dm_exec_query_memory_grants**.

```
--Find all queries waiting in the memory queue
SELECT * FROM sys.dm_exec_query_memory_grants where grant_time is null
```

- Search cache for queries with memory grants usingsys.dm_exec_cached_plans (Transact-SQL) and sys.dm_exec_query_plan (Transact-SQL)

```
-- retrieve every query plan from the plan cache
USE master;
GO
SELECT * FROM sys.dm_exec_cached_plans cp CROSS APPLY sys.dm_exec_query_plan(cp.plan_handle);
GO
```

- Further examine memory-intensive queries using sys.dm_exec_requests.

```
--Find top 5 queries by average CPU time
SELECT TOP 5 total_worker_time/execution_count AS [Avg CPU Time],
Plan_handle, query_plan
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle)
ORDER BY total_worker_time/execution_count DESC;
GO
```

- If a runaway query is suspected, examine the Showplan from sys.dm_exec_query_plan and batch text from sys.dm_exec_sql_text.

  Queries that use dynamic management views that include ORDER BY or aggregates may increase memory consumption and thus contribute to the problem they are troubleshooting.

  The Resource Governor feature enables a database administrator to distribute server resources among resource pools, up to a maximum of 64 pools. Beginning with SQL Server 2008, each pool behaves like a small independent server instance and requires 2 semaphores. The number of rows that are returned from **sys.dm_exec_query_resource_semaphores** can be up to 20 times more than the rows that are returned in SQL Server 2005.

See Also

sys.dm_exec_query_resource_semaphores (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_exec_query_optimizer_info (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns detailed statistics about the operation of the SQL Server query optimizer. You can use this view when tuning a workload to identify query optimization problems or improvements. For example, you can use the total number of optimizations, the elapsed time value, and the final cost value to compare the query optimizations of the current workload and any changes observed during the tuning process. Some counters provide data that is relevant only for SQL Server internal diagnostic use. These counters are marked as "Internal only."

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_exec_query_optimizer_info**.

NAME	DATA TYPE	DESCRIPTION
counter	**nvarchar(4000)**	Name of optimizer statistics event.
occurrence	**bigint**	Number of occurrences of optimization event for this counter.
value	**float**	Average property value per event occurrence.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

sys.dm_exec_query_optimizer_info contains the following properties (counters). All occurrence values are cumulative and are set to 0 at system restart. All values for value fields are set to NULL at system restart. All value-column values that specify an average use the occurrence value from the same row as the denominator in the calculation of the average. All query optimizations are measured when SQL Server determines changes to **dm_exec_query_optimizer_info**, including both user- and system-generated queries. Execution of an already-cached plan does not change values in **dm_exec_query_optimizer_info**, only optimizations are significant.

COUNTER	OCCURRENCE	VALUE
optimizations	Total number of optimizations.	Not applicable

COUNTER	OCCURRENCE	VALUE
elapsed time	Total number of optimizations.	Average elapsed time per optimization of an individual statement (query), in seconds.
final cost	Total number of optimizations.	Average estimated cost for an optimized plan in internal cost units.
trivial plan	Internal only	Internal only
tasks	Internal only	Internal only
no plan	Internal only	Internal only
search 0	Internal only	Internal only
search 0 time	Internal only	Internal only
search 0 tasks	Internal only	Internal only
search 1	Internal only	Internal only
search 1 time	Internal only	Internal only
search 1 tasks	Internal only	Internal only
search 2	Internal only	Internal only
search 2 time	Internal only	Internal only
search 2 tasks	Internal only	Internal only
gain stage 0 to stage 1	Internal only	Internal only
gain stage 1 to stage 2	Internal only	Internal only
timeout	Internal only	Internal only
memory limit exceeded	Internal only	Internal only
insert stmt	Number of optimizations that are for INSERT statements.	Not applicable
delete stmt	Number of optimizations that are for DELETE statements.	Not applicable
update stmt	Number of optimizations that are for UPDATE statements.	Not applicable
contains subquery	Number of optimizations for a query that contains at least one subquery.	Not applicable

COUNTER	OCCURRENCE	VALUE
unnest failed	Internal only	Internal only
tables	Total number of optimizations.	Average number of tables referenced per query optimized.
hints	Number of times some hint was specified. Hints counted include: JOIN, GROUP, UNION and FORCE ORDER query hints, FORCE PLAN set option, and join hints.	Not applicable
order hint	Number of times a force order hint was specified.	Not applicable
join hint	Number of times the join algorithm was forced by a join hint.	Not applicable
view reference	Number of times a view has been referenced in a query.	Not applicable
remote query	Number of optimizations where the query referenced at least one remote data source, such as a table with a four-part name or an OPENROWSET result.	Not applicable
maximum DOP	Total number of optimizations.	Average effective MAXDOP value for an optimized plan. By default, effective MAXDOP is determined by the **max degree of parallelism** server configuration option, and may be overridden for a specific query by the value of the MAXDOP query hint.
maximum recursion level	Number of optimizations in which a MAXRECURSION level greater than 0 has been specified with the query hint.	Average MAXRECURSION level in optimizations where a maximum recursion level is specified with the query hint.
indexed views loaded	Internal only	Internal only
indexed views matched	Number of optimizations where one or more indexed views have been matched.	Average number of views matched.
indexed views used	Number of optimizations where one or more indexed views are used in the output plan after being matched.	Average number of views used.
indexed views updated	Number of optimizations of a DML statement that produce a plan that maintains one or more indexed views.	Average number of views maintained.
dynamic cursor request	Number of optimizations in which a dynamic cursor request has been specified.	Not applicable

COUNTER	OCCURRENCE	VALUE
fast forward cursor request	Number of optimizations in which a fast-forward cursor request has been specified.	Not applicable
merge stmt	Number of optimizations that are for MERGE statements.	Not applicable

Examples

A. Viewing statistics on optimizer execution

What are the current optimizer execution statistics for this instance of SQL Server?

```
SELECT * FROM sys.dm_exec_query_optimizer_info;
```

B. Viewing the total number of optimizations

How many optimizations are performed?

```
SELECT occurrence AS Optimizations FROM sys.dm_exec_query_optimizer_info
WHERE counter = 'optimizations';
```

C. Average elapsed time per optimization

What is the average elapsed time per optimization?

```
SELECT ISNULL(value,0.0) AS ElapsedTimePerOptimization
FROM sys.dm_exec_query_optimizer_info WHERE counter = 'elapsed time';
```

D. Fraction of optimizations that involve subqueries

What fraction of optimized queries contained a subquery?

```
SELECT (SELECT CAST (occurrence AS float) FROM sys.dm_exec_query_optimizer_info WHERE counter = 'contains
subquery') /
       (SELECT CAST (occurrence AS float)
        FROM sys.dm_exec_query_optimizer_info WHERE counter = 'optimizations')
        AS ContainsSubqueryFraction;
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_exec_query_optimizer_memory_gateways (Transact-SQL)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the current status of resource semaphores used to throttle concurrent query optimization.

COLUMN	TYPE	DESCRIPTION
pool_id	**int**	Resource pool ID under Resource Governor
name	**sysname**	Compile gate name (Small Gateway, Medium Gateway, Big Gateway)
max_count	**int**	The maximum configured count of concurrent compiles
active_count	**int**	The currently active count of compiles in this gate
waiter_count	**int**	The number of waiters in this gate
threshold_factor	**bigint**	Threshold factor which defines the maximum memory portion used by query optimization. For the small gateway, threshold_factor indicates the maximum optimizer memory usage in bytes for one query before it is required to gain an access in the small gateway. For the medium and big gateway, threshold_factor shows the portion of total server memory available for this gate. It is used as a divisor when calculating the memory usage threshold for the gate.
threshold	**bigint**	Next threshold memory in bytes. The query is required to gain an access to this gateway if its memory consumption reaches this threshold. “-1” if the query is not required to gain an access to this gateway.
is_active	**bit**	Whether the query is required to pass the current gate or not.

Permissions

SQL Server requires VIEW SERVER STATE permission on the server.

Azure SQL Database requires the VIEW DATABASE STATE permission in the database.

Remarks

SQL Server uses a tiered gateway approach to throttle the number of permitted concurrent compilations. Three gateways are used, including small, medium and big. Gateways help prevent the exhausting of overall memory resources by larger compilation memory-requiring consumers.

Waits on a gateway result in delayed compilation. In addition to delays in compilation, throttled requests will have an associated RESOURCE_SEMAPHORE_QUERY_COMPILE wait type accumulation. The RESOURCE_SEMAPHORE_QUERY_COMPILE wait type may indicate that queries are using a large amount of memory for compilation and that memory has been exhausted, or alternatively there is sufficient memory available overall, however available units in a specific gateway have been exhausted. The output of **sys.dm_exec_query_optimizer_memory_gateways** can be used to troubleshoot scenarios where there was insufficient memory to compile a query execution plan.

Examples

A. Viewing statistics on resource semaphores

What are the current optimizer memory gateway statistics for this instance of SQL Server?

```
SELECT [pool_id], [name], [max_count], [active_count],
       [waiter_count], [threshold_factor], [threshold],
       [is_active]
FROM sys.dm_exec_query_optimizer_memory_gateways;
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)
How to use the DBCC MEMORYSTATUS command to monitor memory usage on SQL Server 2005 Large query compilation waits on RESOURCE_SEMAPHORE_QUERY_COMPILE in SQL Server 2014

sys.dm_exec_query_parallel_workers (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns worker availability information per node.

NAME	DATA TYPE	DESCRIPTION
node_id	**int**	NUMA node ID.
scheduler_count	**int**	Number of schedulers on this node.
max_worker_count	**int**	Maximum number of workers for parallel queries.
reserved_worker_count	**int**	Number of workers reserved by parallel queries, plus number of main workers used by all requests.
free_worker_count	**int**	Number of workers available for tasks. **Note:** every incoming request consumes at least 1 worker, which is subtracted from the free worker count. It is possible that the free worker count can be a negative number on a heavily loaded server.
used_worker_count	**int**	Number of workers used by parallel queries.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

A. Viewing current parallel worker availability

```
SELECT * FROM sys.dm_exec_query_parallel_workers;
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_os_workers (Transact-SQL)

sys.dm_exec_query_plan (Transact-SQL)

5/4/2018 • 6 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the Showplan in XML format for the batch specified by the plan handle. The plan specified by the plan handle can either be cached or currently executing.

The XML schema for the Showplan is published and available at this Microsoft Web site. It is also available in the directory where SQL Server is installed.

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_exec_query_plan ( plan_handle )
```

Arguments

plan_handle
Uniquely identifies a query plan for a batch that is cached or is currently executing.

plan_handle is **varbinary(64)**. *plan_handle* can be obtained from the following dynamic management objects:

sys.dm_exec_cached_plans

sys.dm_exec_query_stats

sys.dm_exec_requests

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
dbid	**smallint**	ID of the context database that was in effect when the Transact-SQL statement corresponding to this plan was compiled. For ad hoc and prepared SQL statements, the ID of the database where the statements were compiled. Column is nullable.
objectid	**int**	ID of the object (for example, stored procedure or user-defined function) for this query plan. For ad hoc and prepared batches, this column is **null**. Column is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
number	**smallint**	Numbered stored procedure integer. For example, a group of procedures for the **orders** application may be named **orderproc;1**, **orderproc;2**, and so on. For ad hoc and prepared batches, this column is **null**. Column is nullable.
encrypted	**bit**	Indicates whether the corresponding stored procedure is encrypted. 0 = not encrypted 1 = encrypted Column is not nullable.
query_plan	**xml**	Contains the compile-time Showplan representation of the query execution plan that is specified with *plan_handle*. The Showplan is in XML format. One plan is generated for each batch that contains, for example ad hoc Transact-SQL statements, stored procedure calls, and user-defined function calls. Column is nullable.

Remarks

Under the following conditions, no Showplan output is returned in the **query_plan** column of the returned table for **sys.dm_exec_query_plan**:

- If the query plan that is specified by using *plan_handle* has been evicted from the plan cache, the **query_plan** column of the returned table is null. For example, this condition may occur if there is a time delay between when the plan handle was captured and when it was used with **sys.dm_exec_query_plan**.

- Some Transact-SQL statements are not cached, such as bulk operation statements or statements containing string literals larger than 8 KB in size. XML Showplans for such statements cannot be retrieved by using **sys.dm_exec_query_plan** unless the batch is currently executing because they do not exist in the cache.

- If a Transact-SQL batch or stored procedure contains a call to a user-defined function or a call to dynamic SQL, for example using EXEC (*string*), the compiled XML Showplan for the user-defined function is not included in the table returned by **sys.dm_exec_query_plan** for the batch or stored procedure. Instead, you must make a separate call to **sys.dm_exec_query_plan** for the plan handle that corresponds to the user-defined function.

  When an ad hoc query uses simple or forced parameterization, the **query_plan** column will contain only the statement text and not the actual query plan. To return the query plan, call **sys.dm_exec_query_plan** for the plan handle of the prepared parameterized query. You can determine whether the query was parameterized by referencing the **sql** column of the sys.syscacheobjects view or the text column of the sys.dm_exec_sql_text dynamic management view.

  Due to a limitation in the number of nested levels allowed in the **xml** data type,

sys.dm_exec_query_plan cannot return query plans that meet or exceed 128 levels of nested elements. In earlier versions of SQL Server, this condition prevented the query plan from returning and generates error 6335. In SQL Server 2005 Service Pack 2 and later versions, the **query_plan** column returns NULL. You can use the sys.dm_exec_text_query_plan (Transact-SQL) dynamic management function to return the output of the query plan in text format.

Permissions

To execute **sys.dm_exec_query_plan**, a user must be a member of the **sysadmin** fixed server role or have the VIEW SERVER STATE permission on the server.

Examples

The following examples show how to use the **sys.dm_exec_query_plan** dynamic management view.

To view the XML Showplans, execute the following queries in the Query Editor of SQL Server Management Studio, then click **ShowPlanXML** in the **query_plan** column of the table returned by **sys.dm_exec_query_plan**. The XML Showplan displays in the Management Studio summary pane. To save the XML Showplan to a file, right-click **ShowPlanXML** in the **query_plan** column, click **Save Results As**, name the file in the format <*file_name*>.sqlplan; for example, MyXMLShowplan.sqlplan.

A. Retrieve the cached query plan for a slow-running Transact-SQL query or batch

Query plans for various types of Transact-SQL batches, such as ad hoc batches, stored procedures, and user-defined functions, are cached in an area of memory called the plan cache. Each cached query plan is identified by a unique identifier called a plan handle. You can specify this plan handle with the **sys.dm_exec_query_plan** dynamic management view to retrieve the execution plan for a particular Transact-SQL query or batch.

If a Transact-SQL query or batch runs a long time on a particular connection to SQL Server, retrieve the execution plan for that query or batch to discover what is causing the delay. The following example shows how to retrieve the XML Showplan for a slow-running query or batch.

> **NOTE**
>
> To run this example, replace the values for *session_id* and *plan_handle* with values specific to your server.

First, retrieve the server process ID (SPID) for the process that is executing the query or batch by using the `sp_who` stored procedure:

```
USE master;
GO
exec sp_who;
GO
```

The result set that is returned by `sp_who` indicates that the SPID is `54`. You can use the SPID with the `sys.dm_exec_requests` dynamic management view to retrieve the plan handle by using the following query:

```
USE master;
GO
SELECT * FROM sys.dm_exec_requests
WHERE session_id = 54;
GO
```

The table that is returned by **sys.dm_exec_requests** indicates that the plan handle for the slow-running query or batch is `0x06000100A27E7C1FA821B10600`, which you can specify as the *plan_handle* argument with `sys.dm_exec_query_plan` to retrieve the execution plan in XML format as follows. The execution plan in XML format for the slow-running query or batch is contained in the **query_plan** column of the table returned by `sys.dm_exec_query_plan`.

```
USE master;
GO
SELECT * FROM sys.dm_exec_query_plan (0x06000100A27E7C1FA821B10600);
GO
```

B. Retrieve every query plan from the plan cache

To retrieve a snapshot of all query plans residing in the plan cache, retrieve the plan handles of all query plans in the cache by querying the `sys.dm_exec_cached_plans` dynamic management view. The plan handles are stored in the `plan_handle` column of `sys.dm_exec_cached_plans`. Then use the CROSS APPLY operator to pass the plan handles to `sys.dm_exec_query_plan` as follows. The XML Showplan output for each plan currently in the plan cache is in the `query_plan` column of the table that is returned.

```
USE master;
GO
SELECT * FROM sys.dm_exec_cached_plans cp CROSS APPLY sys.dm_exec_query_plan(cp.plan_handle);
GO
```

C. Retrieve every query plan for which the server has gathered query statistics from the plan cache

To retrieve a snapshot of all query plans for which the server has gathered statistics that currently reside in the plan cache, retrieve the plan handles of these plans in the cache by querying the `sys.dm_exec_query_stats` dynamic management view. The plan handles are stored in the `plan_handle` column of `sys.dm_exec_query_stats`. Then use the CROSS APPLY operator to pass the plan handles to `sys.dm_exec_query_plan` as follows. The XML Showplan output for each plan for which the server has gathered statistics currently in the plan cache is in the `query_plan` column of the table that is returned.

```
USE master;
GO
SELECT * FROM sys.dm_exec_query_stats qs CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle);
GO
```

D. Retrieve information about the top five queries by average CPU time

The following example returns the plans and average CPU time for the top five queries.

```
SELECT TOP 5 total_worker_time/execution_count AS [Avg CPU Time],
Plan_handle, query_plan
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle)
ORDER BY total_worker_time/execution_count DESC;
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_cached_plans (Transact-SQL)
sys.dm_exec_query_stats (Transact-SQL)
sys.dm_exec_requests (Transact-SQL)
sp_who (Transact-SQL)
Showplan Logical and Physical Operators Reference

sys.dm_exec_text_query_plan (Transact-SQL)

sys.dm_exec_query_profiles (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Monitors real time query progress while the query is in execution. For example, use this DMV to determine which part of the query is running slow. Join this DMV with other system DMVs using the columns identified in the description field. Or, join this DMV with other performance counters (such as Performance Monitor, xperf) by using the timestamp columns.

Table Returned

The counters returned are per operator per thread. The results are dynamic and do not match the results of existing options such as SET STATISTICS XML ON which only create output when the query is finished.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**smallint**	Identifies the session in which this query runs. References dm_exec_sessions.session_id.
request_id	**int**	Identifies the target request. References dm_exec_sessions.request_id.
sql_handle	**varbinary(64)**	Identifies the target query. References dm_exec_query_stats.sql_handle.
plan_handle	**varbinary(64)**	Identify target query (references dm_exec_query_stats.plan_handle.
physical_operator_name	**nvarchar(256)**	Physical operator name.
node_id	**int**	Identifies an operator node in the query tree.
thread_id	**int**	Distinguishes the threads (for a parallel query) belonging to the same query operator node.
task_address	**varbinary(8)**	Identifies the SQLOS task that this thread is using. References dm_os_tasks.task_address.
row_count	**bigint**	Number of rows returned by the operator so far.
rewind_count	**bigint**	Number of rewinds so far.
rebind_count	**bigint**	Number of rebinds so far.
end_of_scan_count	**bigint**	Number of end of scans so far.

COLUMN NAME	DATA TYPE	DESCRIPTION
estimate_row_count	**bigint**	Estimated number of rows. It can be useful to compare to estimated_row_count to the actual row_count.
first_active_time	**bigint**	The time, in milliseconds, when the operator was first called.
last_active_time	**bigint**	The time, in milliseconds, when the operator was last called.
open_time	**bigint**	Timestamp when open (in milliseconds).
first_row_time	**bigint**	Timestamp when first row was opened (in milliseconds).
last_row_time	**bigint**	Timestamp when last row was opened(in milliseconds).
close_time	**bigint**	Timestamp when close (in milliseconds).
elapsed_time_ms	**bigint**	Total elapsed time (in milliseconds) used by the target node's operations so far.
cpu_time_ms	**bigint**	Total CPU time (in milliseconds) use by target node's operations so far.
database_id	**smallint**	ID of the database that contains the object on which the reads and writes are being performed.
object_id	**int**	The identifier for the object on which the reads and writes are being performed. References sys.objects.object_id.
index_id	**int**	The index (if any) the rowset is opened against.
scan_count	**bigint**	Number of table/index scans so far.
logical_read_count	**bigint**	Number of logical reads so far.
physical_read_count	**bigint**	Number of physical reads so far.
read_ahead_count	**bigint**	Number of read-aheads so far.
write_page_count	**bigint**	Number of page-writes so far due to spilling.
lob_logical_read_count	**bigint**	Number of LOB logical reads so far.
lob_physical_read_count	**bigint**	Number of LOB physical reads so far.

COLUMN NAME	DATA TYPE	DESCRIPTION
lob_read_ahead_count	**bigint**	Number of LOB read-aheads so far.
segment_read_count	**int**	Number of segment read-aheads so far.
segment_skip_count	**int**	Number of segments skipped so far.
actual_read_row_count	**bigint**	Number of rows read by an operator before the residual predicate was applied.
estimated_read_row_count	**bigint**	**Applies to:** Beginning with SQL Server 2016 (13.x) SP1. Number of rows estimated to be read by an operator before the residual predicate was applied.

General Remarks

If the query plan node does not have any IO, all the IO-related counters are set to NULL.

The IO-related counters reported by this DMV are more granular than the ones reported by SET STATISTICS IO in the following two ways:

- SET STATISTICS IO groups the counters for all IO to a given table together. With this DMV you will get separate counters for every node in the query plan that performs IO to the table.
- If there is a parallel scan, this DMV reports counters for each of the parallel threads working on the scan.

  Starting with SQL Server 2016 (13.x) SP1, the standard query execution statistics profiling infrastructure exists side-by-side with a lightweight query execution statistics profiling infrastructure. The new query execution statistics profiling infrastructure dramatically reduces performance overhead of collecting per-operator query execution statistics, such as actual number of rows. This feature can be enabled either using global startup trace flag 7412, or is automatically turned on when query_thread_profile extended event is used.

> **NOTE**
> CPU and elapsed times are not supported under the lightweight query execution statistics profiling infrastructure to reduce performance impact.

SET STATISTICS XML ON and SET STATISTICS PROFILE ON always use the legacy query execution statistics profiling infrastructure.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

Step 1: Login to a session in which you plan to run the query you will analyze with sys.dm_exec_query_profiles. To configure the query for profiling use SET STATISTICS PROFILE ON. Run your query in this same session.

```
--Configure query for profiling with sys.dm_exec_query_profiles
SET STATISTICS PROFILE ON;
GO

--Or enable query profiling globally under SQL Server 2016 SP1 or above
DBCC TRACEON (7412, -1);
GO

--Next, run your query in this session, or in any other session if query profiling has been enabled globally
```

Step 2: Login to a second session that is different from the session in which your query is running.

The following statement summarizes the progress made by the query currently running in session 54. To do this, it calculates the total number of output rows from all threads for each node, and compares it to the estimated number of output rows for that node.

```
--Run this in a different session than the session in which your query is running.
--Note that you may need to change session id 54 below with the session id you want to monitor.
SELECT node_id,physical_operator_name, SUM(row_count) row_count,
  SUM(estimate_row_count) AS estimate_row_count,
  CAST(SUM(row_count)*100 AS float)/SUM(estimate_row_count)
FROM sys.dm_exec_query_profiles
WHERE session_id=54
GROUP BY node_id,physical_operator_name
ORDER BY node_id;
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_exec_query_resource_semaphores (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the information about the current query-resource semaphore status in SQL Server. **sys.dm_exec_query_resource_semaphores** provides general query-execution memory status and allows you to determine whether the system can access enough memory. This view complements memory information obtained from sys.dm_os_memory_clerks to provide a complete picture of server memory status. **sys.dm_exec_query_resource_semaphores** returns one row for the regular resource semaphore and another row for the small-query resource semaphore. There are two requirements for a small-query semaphore:

- The memory grant requested should be less than 5 MB
- The query cost should be less than 3 cost units

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_exec_query_resource_semaphores**.

COLUMN NAME	DATA TYPE	DESCRIPTION
resource_semaphore_id	**smallint**	Nonunique ID of the resource semaphore. 0 for the regular resource semaphore and 1 for the small-query resource semaphore.
target_memory_kb	**bigint**	Grant usage target in kilobytes.
max_target_memory_kb	**bigint**	Maximum potential target in kilobytes. NULL for the small-query resource semaphore.
total_memory_kb	**bigint**	Memory held by the resource semaphore in kilobytes. If the system is under memory pressure or if forced minimum memory is granted frequently, this value can be larger than the **target_memory_kb** or **max_target_memory_kb** values. Total memory is a sum of available and granted memory.
available_memory_kb	**bigint**	Memory available for a new grant in kilobytes.
granted_memory_kb	**bigint**	Total granted memory in kilobytes.

COLUMN NAME	DATA TYPE	DESCRIPTION
used_memory_kb	**bigint**	Physically used part of granted memory in kilobytes.
grantee_count	**int**	Number of active queries that have their grants satisfied.
waiter_count	**int**	Number of queries waiting for grants to be satisfied.
timeout_error_count	**bigint**	Total number of time-out errors since server startup. NULL for the small-query resource semaphore.
forced_grant_count	**bigint**	Total number of forced minimum-memory grants since server startup. NULL for the small-query resource semaphore.
pool_id	**int**	ID of the resource pool to which this resource semaphore belongs.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

Queries that use dynamic management views that include ORDER BY or aggregates might increase memory consumption and thus contribute to the problem they are troubleshooting.

Use **sys.dm_exec_query_resource_semaphores** for troubleshooting but do not include it in applications that will use future versions of SQL Server.

The Resource Governor feature enables a database administrator to distribute server resources among resource pools, up to a maximum of 64 pools. In SQL Server 2012 (11.x) and higher, each pool behaves like a small independent server instance and requires 2 semaphores.

See Also

Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_query_memory_grants (Transact-SQL)

sys.dm_exec_query_statistics_xml (Transact-SQL)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns query execution plan for in-flight requests. Use this DMV to retrieve showplan XML with transient statistics.

Syntax

```
sys.dm_exec_query_statistics_xml(session_id)
```

Arguments

session_id
Is the session id executing the batch to be looked up. *session_id* is **smallint**. *session_id* can be obtained from the following dynamic management objects:

- sys.dm_exec_requests
- sys.dm_exec_sessions
- sys.dm_exec_connections

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**smallint**	ID of the session. Not nullable.
request_id	**int**	ID of the request. Not nullable.
sql_handle	**varbinary(64)**	Hash map of SQL text of the request. Nullable.
plan_handle	**varbinary(64)**	Hash map of query plan. Nullable.
query_plan	**xml**	Showplan XML with partial statistics. Nullable.

Remarks

This system function is available starting with SQL Server 2016 (13.x) SP1.

This system function works under both **standard** and **lightweight** query execution statistics profiling infrastructure.

Standard statistics profiling infrastructure can be enabled by using:

- SET STATISTICS XML ON

- SET STATISTICS PROFILE ON
- the `query_post_execution_showplan` extended event.

Lightweight statistics profiling infrastructure is available in SQL Server 2014 (12.x) SP2 and SQL Server 2016 (13.x) and can be enabled:

- Globally by using trace flag 7412.
- Using the *query_thread_profile* extended event.

> **NOTE**
>
> Once enabled by trace flag 7412, lightweight profiling will be enabled to any consumer of the query execution statistics profiling infrastructure instead of standard profiling, such as the DMV sys.dm_exec_query_profiles. However, standard profiling is still used for SET STATISTICS XML, *Include Actual Plan* action in Management Studio, and `query_post_execution_showplan` xEvent.

> **IMPORTANT**
>
> In TPC-C like workload tests, enabling the lightweight statistics profiling infrastructure adds a 1.5 to 2 percent overhead. In contrast, the standard statistics profiling infrastructure can add up to 90 percent overhead for the same workload scenario.

Permissions

Requires `VIEW SERVER STATE` permission on the server.

Examples

A. Looking at live query plan and execution statistics for a running batch

The following example queries **sys.dm_exec_requests** to find the interesting query and copy its `session_id` from the output.

```
SELECT * FROM sys.dm_exec_requests;
GO
```

Then, to obtain the live query plan and execution statistics, use the copied `session_id` with system function **sys.dm_exec_query_statistics_xml**.

```
--Run this in a different session than the session in which your query is running.
SELECT * FROM sys.dm_exec_query_statistics_xml(< copied session_id >);
GO
```

Or combined for all running requests.

```
--Run this in a different session than the session in which your query is running.
SELECT * FROM sys.dm_exec_requests
CROSS APPLY sys.dm_exec_query_statistics_xml(session_id);
GO
```

See Also

Trace Flags

Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_query_stats (Transact-SQL)

5/4/2018 • 14 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns aggregate performance statistics for cached query plans in SQL Server. The view contains one row per query statement within the cached plan, and the lifetime of the rows are tied to the plan itself. When a plan is removed from the cache, the corresponding rows are eliminated from this view.

> **NOTE**
> An initial query of **sys.dm_exec_query_stats** might produce inaccurate results if there is a workload currently executing on the server. More accurate results may be determined by rerunning the query.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_exec_query_stats**.

COLUMN NAME	DATA TYPE	DESCRIPTION
sql_handle	**varbinary(64)**	Is a token that refers to the batch or stored procedure that the query is part of. **sql_handle**, together with **statement_start_offset** and **statement_end_offset**, can be used to retrieve the SQL text of the query by calling the **sys.dm_exec_sql_text** dynamic management function.
statement_start_offset	**int**	Indicates, in bytes, beginning with 0, the starting position of the query that the row describes within the text of its batch or persisted object.
statement_end_offset	**int**	Indicates, in bytes, starting with 0, the ending position of the query that the row describes within the text of its batch or persisted object. For versions before SQL Server 2014 (12.x), a value of -1 indicates the end of the batch. Trailing comments are no longer include.
plan_generation_num	**bigint**	A sequence number that can be used to distinguish between instances of plans after a recompile.

COLUMN NAME	DATA TYPE	DESCRIPTION
plan_handle	**varbinary(64)**	A token that refers to the compiled plan that the query is part of. This value can be passed to the sys.dm_exec_query_plan dynamic management function to obtain the query plan. Will always be 0x000 when a natively compiled stored procedure queries a memory-optimized table.
creation_time	**datetime**	Time at which the plan was compiled.
last_execution_time	**datetime**	Last time at which the plan started executing.
execution_count	**bigint**	Number of times that the plan has been executed since it was last compiled.
total_worker_time	**bigint**	Total amount of CPU time, reported in microseconds (but only accurate to milliseconds), that was consumed by executions of this plan since it was compiled. For natively compiled stored procedures, **total_worker_time** may not be accurate if many executions take less than 1 millisecond.
last_worker_time	**bigint**	CPU time, reported in microseconds (but only accurate to milliseconds), that was consumed the last time the plan was executed. [1]
min_worker_time	**bigint**	Minimum CPU time, reported in microseconds (but only accurate to milliseconds), that this plan has ever consumed during a single execution. [1]
max_worker_time	**bigint**	Maximum CPU time, reported in microseconds (but only accurate to milliseconds), that this plan has ever consumed during a single execution. [1]
total_physical_reads	**bigint**	Total number of physical reads performed by executions of this plan since it was compiled. Will always be 0 querying a memory-optimized table.

COLUMN NAME	DATA TYPE	DESCRIPTION
last_physical_reads	**bigint**	Number of physical reads performed the last time the plan was executed. Will always be 0 querying a memory-optimized table.
min_physical_reads	**bigint**	Minimum number of physical reads that this plan has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
max_physical_reads	**bigint**	Maximum number of physical reads that this plan has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
total_logical_writes	**bigint**	Total number of logical writes performed by executions of this plan since it was compiled. Will always be 0 querying a memory-optimized table.
last_logical_writes	**bigint**	Number of the number of buffer pool pages dirtied the last time the plan was executed. If a page is already dirty (modified) no writes are counted. Will always be 0 querying a memory-optimized table.
min_logical_writes	**bigint**	Minimum number of logical writes that this plan has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
max_logical_writes	**bigint**	Maximum number of logical writes that this plan has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
total_logical_reads	**bigint**	Total number of logical reads performed by executions of this plan since it was compiled. Will always be 0 querying a memory-optimized table.

COLUMN NAME	DATA TYPE	DESCRIPTION
last_logical_reads	**bigint**	Number of logical reads performed the last time the plan was executed. Will always be 0 querying a memory-optimized table.
min_logical_reads	**bigint**	Minimum number of logical reads that this plan has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
max_logical_reads	**bigint**	Maximum number of logical reads that this plan has ever performed during a single execution. Will always be 0 querying a memory-optimized table.
total_clr_time	**bigint**	Time, reported in microseconds (but only accurate to milliseconds), consumed inside Microsoft .NET Framework common language runtime (CLR) objects by executions of this plan since it was compiled. The CLR objects can be stored procedures, functions, triggers, types, and aggregates.
last_clr_time	**bigint**	Time, reported in microseconds (but only accurate to milliseconds) consumed by execution inside .NET Framework CLR objects during the last execution of this plan. The CLR objects can be stored procedures, functions, triggers, types, and aggregates.
min_clr_time	**bigint**	Minimum time, reported in microseconds (but only accurate to milliseconds), that this plan has ever consumed inside .NET Framework CLR objects during a single execution. The CLR objects can be stored procedures, functions, triggers, types, and aggregates.
max_clr_time	**bigint**	Maximum time, reported in microseconds (but only accurate to milliseconds), that this plan has ever consumed inside the .NET Framework CLR during a single execution. The CLR objects can be stored procedures, functions, triggers, types, and aggregates.

COLUMN NAME	DATA TYPE	DESCRIPTION
total_elapsed_time	**bigint**	Total elapsed time, reported in microseconds (but only accurate to milliseconds), for completed executions of this plan.
last_elapsed_time	**bigint**	Elapsed time, reported in microseconds (but only accurate to milliseconds), for the most recently completed execution of this plan.
min_elapsed_time	**bigint**	Minimum elapsed time, reported in microseconds (but only accurate to milliseconds), for any completed execution of this plan.
max_elapsed_time	**bigint**	Maximum elapsed time, reported in microseconds (but only accurate to milliseconds), for any completed execution of this plan.
query_hash	**Binary(8)**	Binary hash value calculated on the query and used to identify queries with similar logic. You can use the query hash to determine the aggregate resource usage for queries that differ only by literal values.
query_plan_hash	**binary(8)**	Binary hash value calculated on the query execution plan and used to identify similar query execution plans. You can use query plan hash to find the cumulative cost of queries with similar execution plans. Will always be 0x000 when a natively compiled stored procedure queries a memory-optimized table.
total_rows	**bigint**	Total number of rows returned by the query. Cannot be null. Will always be 0 when a natively compiled stored procedure queries a memory-optimized table.
last_rows	**bigint**	Number of rows returned by the last execution of the query. Cannot be null. Will always be 0 when a natively compiled stored procedure queries a memory-optimized table.

COLUMN NAME	DATA TYPE	DESCRIPTION
min_rows	**bigint**	Minimum number of rows ever returned by the query during one execution. Cannot be null. Will always be 0 when a natively compiled stored procedure queries a memory-optimized table.
max_rows	**bigint**	Maximum number of rows ever returned by the query during one execution. Cannot be null. Will always be 0 when a natively compiled stored procedure queries a memory-optimized table.
statement_sql_handle	**varbinary(64)**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. Populated with non-NULL values only if Query Store is turned on and collecting the stats for that particular query.
statement_context_id	**bigint**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. Populated with non-NULL values only if Query Store is turned on and collecting the stats for that particular query.
total_dop	**bigint**	The total sum of degree of parallelism this plan used since it was compiled. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
last_dop	**bigint**	The degree of parallelism when this plan executed last time. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
min_dop	**bigint**	The minimum degree of parallelism this plan ever used during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

COLUMN NAME	DATA TYPE	DESCRIPTION
max_dop	**bigint**	The maximum degree of parallelism this plan ever used during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
total_grant_kb	**bigint**	The total amount of reserved memory grant in KB this plan received since it was compiled. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
last_grant_kb	**bigint**	The amount of reserved memory grant in KB when this plan executed last time. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
min_grant_kb	**bigint**	The minimum amount of reserved memory grant in KB this plan ever received during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
max_grant_kb	**bigint**	The maximum amount of reserved memory grant in KB this plan ever received during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
total_used_grant_kb	**bigint**	The total amount of reserved memory grant in KB this plan used since it was compiled. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
last_used_grant_kb	**bigint**	The amount of used memory grant in KB when this plan executed last time. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

COLUMN NAME	DATA TYPE	DESCRIPTION
min_used_grant_kb	**bigint**	The minimum amount of used memory grant in KB this plan ever used during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
max_used_grant_kb	**bigint**	The maximum amount of used memory grant in KB this plan ever used during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
total_ideal_grant_kb	**bigint**	The total amount of ideal memory grant in KB this plan estimated since it was compiled. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
last_ideal_grant_kb	**bigint**	The amount of ideal memory grant in KB when this plan executed last time. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
min_ideal_grant_kb	**bigint**	The minimum amount of ideal memory grant in KB this plan ever estimated during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
max_ideal_grant_kb	**bigint**	The maximum amount of ideal memory grant in KB this plan ever estimated during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
total_reserved_threads	**bigint**	The total sum of reserved parallel threads this plan ever used since it was compiled. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

COLUMN NAME	DATA TYPE	DESCRIPTION
last_reserved_threads	**bigint**	The number of reserved parallel threads when this plan executed last time. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
min_reserved_threads	**bigint**	The minimum number of reserved parallel threads this plan ever used during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
max_reserved_threads	**bigint**	The maximum number of reserved parallel threads this plan ever used during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
total_used_threads	**bigint**	The total sum of used parallel threads this plan ever used since it was compiled. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
last_used_threads	**bigint**	The number of used parallel threads when this plan executed last time. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
min_used_threads	**bigint**	The minimum number of used parallel threads this plan ever used during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
max_used_threads	**bigint**	The maximum number of used parallel threads this plan ever used during one execution. It will always be 0 for querying a memory-optimized table. **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

COLUMN NAME	DATA TYPE	DESCRIPTION
total_columnstore_segment_reads	**bigint**	The total sum of columnstore segments read by the query. Cannot be null. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
last_columnstore_segment_reads	**bigint**	The number of columnstore segments read by the last execution of the query. Cannot be null. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
min_columnstore_segment_reads	**bigint**	The minimum number of columnstore segments ever read by the query during one execution. Cannot be null. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
max_columnstore_segment_reads	**bigint**	The maximum number of columnstore segments ever read by the query during one execution. Cannot be null. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
total_columnstore_segment_skips	**bigint**	The total sum of columnstore segments skipped by the query. Cannot be null. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
last_columnstore_segment_skips	**bigint**	The number of columnstore segments skipped by the last execution of the query. Cannot be null. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
min_columnstore_segment_skips	**bigint**	The minimum number of columnstore segments ever skipped by the query during one execution. Cannot be null. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3

COLUMN NAME	DATA TYPE	DESCRIPTION
max_columnstore_segment_skips	**bigint**	The maximum number of columnstore segments ever skipped by the query during one execution. Cannot be null. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
total_spills	**bigint**	The total number of pages spilled by execution of this query since it was compiled. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
last_spills	**bigint**	The number of pages spilled the last time the query was executed. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
min_spills	**bigint**	The minimum number of pages that this query has ever spilled during a single execution. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
max_spills	**bigint**	The maximum number of pages that this query has ever spilled during a single execution. **Applies to**: Starting with SQL Server 2016 (13.x) SP2 and SQL Server 2017 (14.x) CU3
pdw_node_id	**int**	The identifier for the node that this distribution is on. **Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse

> **NOTE**
>
> [1] For natively compiled stored procedures when statistics collection is enabled, worker time is collected in milliseconds. If the query executes in less than one millisecond, the value will be 0.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

Statistics in the view are updated when a query is completed.

Examples

A. Finding the TOP N queries

The following example returns information about the top five queries ranked by average CPU time. This example aggregates the queries according to their query hash so that logically equivalent queries are grouped by their cumulative resource consumption.

```
SELECT TOP 5 query_stats.query_hash AS "Query Hash",
    SUM(query_stats.total_worker_time) / SUM(query_stats.execution_count) AS "Avg CPU Time",
    MIN(query_stats.statement_text) AS "Statement Text"
FROM
    (SELECT QS.*,
    SUBSTRING(ST.text, (QS.statement_start_offset/2) + 1,
    ((CASE statement_end_offset
        WHEN -1 THEN DATALENGTH(ST.text)
        ELSE QS.statement_end_offset END
            - QS.statement_start_offset)/2) + 1) AS statement_text
     FROM sys.dm_exec_query_stats AS QS
     CROSS APPLY sys.dm_exec_sql_text(QS.sql_handle) as ST) as query_stats
GROUP BY query_stats.query_hash
ORDER BY 2 DESC;
```

B. Returning row count aggregates for a query

The following example returns row count aggregate information (total rows, minimum rows, maximum rows and last rows) for queries.

```
SELECT qs.execution_count,
    SUBSTRING(qt.text,qs.statement_start_offset/2 +1,
                 (CASE WHEN qs.statement_end_offset = -1
                       THEN LEN(CONVERT(nvarchar(max), qt.text)) * 2
                       ELSE qs.statement_end_offset end -
                            qs.statement_start_offset
                 )/2
             ) AS query_text,
     qt.dbid, dbname= DB_NAME (qt.dbid), qt.objectid,
     qs.total_rows, qs.last_rows, qs.min_rows, qs.max_rows
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS qt
WHERE qt.text like '%SELECT%'
ORDER BY qs.execution_count DESC;
```

See also

Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_sql_text (Transact-SQL)
sys.dm_exec_query_plan (Transact-SQL)
sys.dm_exec_procedure_stats (Transact-SQL)
sys.dm_exec_trigger_stats (Transact-SQL)
sys.dm_exec_cached_plans (Transact-SQL)

sys.dm_exec_requests (Transact-SQL)

5/4/2018 • 7 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about each request that is executing within SQL Server.

> **NOTE**
> To execute code that is outside SQL Server (for example, extended stored procedures and distributed queries), a thread has to execute outside the control of the non-preemptive scheduler. To do this, a worker switches to preemptive mode. Time values returned by this dynamic management view do not include time spent in preemptive mode.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**smallint**	ID of the session to which this request is related. Is not nullable.
request_id	**int**	ID of the request. Unique in the context of the session. Is not nullable.
start_time	**datetime**	Timestamp when the request arrived. Is not nullable.
status	**nvarchar(30)**	Status of the request. This can be one of the following: Background Running Runnable Sleeping Suspended Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
command	**nvarchar(32)**	Identifies the current type of command that is being processed. Common command types include the following: SELECT INSERT UPDATE DELETE BACKUP LOG BACKUP DATABASE DBCC FOR The text of the request can be retrieved by using sys.dm_exec_sql_text with the corresponding sql_handle for the request. Internal system processes set the command based on the type of task they perform. Tasks can include the following: LOCK MONITOR CHECKPOINTLAZY WRITER Is not nullable.
sql_handle	**varbinary(64)**	Hash map of the SQL text of the request. Is nullable.
statement_start_offset	**int**	Number of characters into the currently executing batch or stored procedure at which the currently executing statement starts. Can be used together with the sql_handle, the statement_end_offset, and the sys.dm_exec_sql_text dynamic management function to retrieve the currently executing statement for the request. Is nullable.
statement_end_offset	**int**	Number of characters into the currently executing batch or stored procedure at which the currently executing statement ends. Can be used together with the sql_handle, the statement_end_offset, and the sys.dm_exec_sql_text dynamic management function to retrieve the currently executing statement for the request. Is nullable.
plan_handle	**varbinary(64)**	Hash map of the plan for SQL execution. Is nullable.
database_id	**smallint**	ID of the database the request is executing against. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
user_id	**int**	ID of the user who submitted the request. Is not nullable.
connection_id	**uniqueidentifier**	ID of the connection on which the request arrived. Is nullable.
blocking_session_id	**smallint**	ID of the session that is blocking the request. If this column is NULL, the request is not blocked, or the session information of the blocking session is not available (or cannot be identified). -2 = The blocking resource is owned by an orphaned distributed transaction. -3 = The blocking resource is owned by a deferred recovery transaction. -4 = Session ID of the blocking latch owner could not be determined at this time because of internal latch state transitions.
wait_type	**nvarchar(60)**	If the request is currently blocked, this column returns the type of wait. Is nullable. For information about types of waits, see sys.dm_os_wait_stats (Transact-SQL).
wait_time	**int**	If the request is currently blocked, this column returns the duration in milliseconds, of the current wait. Is not nullable.
last_wait_type	**nvarchar(60)**	If this request has previously been blocked, this column returns the type of the last wait. Is not nullable.
wait_resource	**nvarchar(256)**	If the request is currently blocked, this column returns the resource for which the request is currently waiting. Is not nullable.
open_transaction_count	**int**	Number of transactions that are open for this request. Is not nullable.
open_resultset_count	**int**	Number of result sets that are open for this request. Is not nullable.
transaction_id	**bigint**	ID of the transaction in which this request executes. Is not nullable.
context_info	**varbinary(128)**	CONTEXT_INFO value of the session. Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
percent_complete	**real**	Percentage of work completed for the following commands: ALTER INDEX REORGANIZE AUTO_SHRINK option with ALTER DATABASE BACKUP DATABASE DBCC CHECKDB DBCC CHECKFILEGROUP DBCC CHECKTABLE DBCC INDEXDEFRAG DBCC SHRINKDATABASE DBCC SHRINKFILE RECOVERY RESTORE DATABASE ROLLBACK TDE ENCRYPTION Is not nullable.
estimated_completion_time	**bigint**	Internal only. Is not nullable.
cpu_time	**int**	CPU time in milliseconds that is used by the request. Is not nullable.
total_elapsed_time	**int**	Total time elapsed in milliseconds since the request arrived. Is not nullable.
scheduler_id	**int**	ID of the scheduler that is scheduling this request. Is not nullable.
task_address	**varbinary(8)**	Memory address allocated to the task that is associated with this request. Is nullable.
reads	**bigint**	Number of reads performed by this request. Is not nullable.
writes	**bigint**	Number of writes performed by this request. Is not nullable.
logical_reads	**bigint**	Number of logical reads that have been performed by the request. Is not nullable.
text_size	**int**	TEXTSIZE setting for this request. Is not nullable.
language	**nvarchar(128)**	Language setting for the request. Is nullable.
date_format	**nvarchar(3)**	DATEFORMAT setting for the request. Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
date_first	**smallint**	DATEFIRST setting for the request. Is not nullable.
quoted_identifier	**bit**	1 = QUOTED_IDENTIFIER is ON for the request. Otherwise, it is 0. Is not nullable.
arithabort	**bit**	1 = ARITHABORT setting is ON for the request. Otherwise, it is 0. Is not nullable.
ansi_null_dflt_on	**bit**	1 = ANSI_NULL_DFLT_ON setting is ON for the request. Otherwise, it is 0. Is not nullable.
ansi_defaults	**bit**	1 = ANSI_DEFAULTS setting is ON for the request. Otherwise, it is 0. Is not nullable.
ansi_warnings	**bit**	1 = ANSI_WARNINGS setting is ON for the request. Otherwise, it is 0. Is not nullable.
ansi_padding	**bit**	1 = ANSI_PADDING setting is ON for the request. Otherwise, it is 0. Is not nullable.
ansi_nulls	**bit**	1 = ANSI_NULLS setting is ON for the request. Otherwise, it is 0. Is not nullable.
concat_null_yields_null	**bit**	1 = CONCAT_NULL_YIELDS_NULL setting is ON for the request. Otherwise, it is 0. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_isolation_level	**smallint**	Isolation level with which the transaction for this request is created. Is not nullable. 0 = Unspecified 1 = ReadUncomitted 2 = ReadCommitted 3 = Repeatable 4 = Serializable 5 = Snapshot
lock_timeout	**int**	Lock time-out period in milliseconds for this request. Is not nullable.
deadlock_priority	**int**	DEADLOCK_PRIORITY setting for the request. Is not nullable.
row_count	**bigint**	Number of rows that have been returned to the client by this request. Is not nullable.
prev_error	**int**	Last error that occurred during the execution of the request. Is not nullable.
nest_level	**int**	Current nesting level of code that is executing on the request. Is not nullable.
granted_query_memory	**int**	Number of pages allocated to the execution of a query on the request. Is not nullable.
executing_managed_code	**bit**	Indicates whether a specific request is currently executing common language runtime objects, such as routines, types, and triggers. It is set for the full time a common language runtime object is on the stack, even while running Transact-SQL from within common language runtime. Is not nullable.
group_id	**int**	ID of the workload group to which this query belongs. Is not nullable.
query_hash	**binary(8)**	Binary hash value calculated on the query and used to identify queries with similar logic. You can use the query hash to determine the aggregate resource usage for queries that differ only by literal values.

COLUMN NAME	DATA TYPE	DESCRIPTION
query_plan_hash	**binary(8)**	Binary hash value calculated on the query execution plan and used to identify similar query execution plans. You can use query plan hash to find the cumulative cost of queries with similar execution plans.
statement_sql_handle	**varbinary(64)**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. SQL handle of the individual query.
statement_context_id	**bigint**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The optional foreign key to sys.query_context_settings.
dop	**int**	**Applies to**: SQL Server 2016 (13.x) through SQL Server 2017. The degree of parallelism of the query.
parallel_worker_count	**int**	**Applies to**: SQL Server 2016 (13.x) through SQL Server 2017. The number of reserved parallel workers if this is a parallel query.
external_script_request_id	**uniqueidentifier**	**Applies to**: SQL Server 2016 (13.x) through SQL Server 2017. The external script request ID associated with the current request.
is_resumable	**bit**	**Applies to**: SQL Server 2017 (14.x) through SQL Server 2017. Indicates whether the request is a resumable index operation.

Permissions

If the user has `VIEW SERVER STATE` permission on the server, the user will see all executing sessions on the instance of SQL Server; otherwise, the user will see only the current session. `VIEW SERVER STATE` cannot be granted in SQL Database so `sys.dm_exec_requests` is always limited to the current connection.

Examples

A. Finding the query text for a running batch

The following example queries `sys.dm_exec_requests` to find the interesting query and copy its `sql_handle` from the output.

```
SELECT * FROM sys.dm_exec_requests;
GO
```

Then, to obtain the statement text, use the copied `sql_handle` with system function `sys.dm_exec_sql_text(sql_handle)`.

```
SELECT * FROM sys.dm_exec_sql_text(< copied sql_handle >);
GO
```

B. Finding all locks that a running batch is holding

The following example queries **sys.dm_exec_requests** to find the interesting batch and copy its `transaction_id` from the output.

```
SELECT * FROM sys.dm_exec_requests;
GO
```

Then, to find lock information, use the copied `transaction_id` with the system function **sys.dm_tran_locks**.

```
SELECT * FROM sys.dm_tran_locks
WHERE request_owner_type = N'TRANSACTION'
    AND request_owner_id = < copied transaction_id >;
GO
```

C. Finding all currently blocked requests

The following example queries **sys.dm_exec_requests** to find information about blocked requests.

```
SELECT session_id ,status ,blocking_session_id
    ,wait_type ,wait_time ,wait_resource
    ,transaction_id
FROM sys.dm_exec_requests
WHERE status = N'suspended';
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_os_memory_clerks (Transact-SQL)
sys.dm_os_sys_info (Transact-SQL)
sys.dm_exec_query_memory_grants (Transact-SQL)
sys.dm_exec_query_plan (Transact-SQL)
sys.dm_exec_sql_text (Transact-SQL)

sys.dm_exec_session_wait_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about all the waits encountered by threads that executed for each session. You can use this view to diagnose performance issues with the SQL Server session and also with specific queries and batches. This view returns session the same information that is aggregated for sys.dm_os_wait_stats (Transact-SQL) but provides the **session_id** number as well.

Applies to: SQL Server (SQL Server 2016 (13.x) through SQL Server 2017).

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**smallint**	The id of the session.
wait_type	**nvarchar(60)**	Name of the wait type. For more information, see sys.dm_os_wait_stats (Transact-SQL).
waiting_tasks_count	**bigint**	Number of waits on this wait type. This counter is incremented at the start of each wait.
wait_time_ms	**bigint**	Total wait time for this wait type in milliseconds. This time is inclusive of signal_wait_time_ms.
max_wait_time_ms	**bigint**	Maximum wait time on this wait type.
signal_wait_time_ms	**bigint**	Difference between the time that the waiting thread was signaled and when it started running.

Remarks

This DMV resets the information for a session when the session is opened, or when the session is reset (if connection pooling),

For information about the wait types, see sys.dm_os_wait_stats (Transact-SQL).

Permissions

If the user has **VIEW SERVER STATE** permission on the server, the user will see all executing sessions on the instance of SQL Server; otherwise, the user will see only the current session.

See Also

Dynamic Management Views and Functions (Transact-SQL)
SQL Server Operating System Related Dynamic Management Views (Transact-SQL)
sys.dm_os_wait_stats (Transact-SQL)

sys.dm_exec_sessions (Transact-SQL)

5/4/2018 • 7 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row per authenticated session on SQL Server. sys.dm_exec_sessions is a server-scope view that shows information about all active user connections and internal tasks. This information includes client version, client program name, client login time, login user, current session setting, and more. Use sys.dm_exec_sessions to first view the current system load and to identify a session of interest, and then learn more information about that session by using other dynamic management views or dynamic management functions.

The sys.dm_exec_connections, sys.dm_exec_sessions, and sys.dm_exec_requests dynamic management views map to the sys.sysprocesses system table.

> **NOTE:** To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_exec_sessions**.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC INFORMATION
session_id	**smallint**	Identifies the session associated with each active primary connection. Is not nullable.
login_time	**datetime**	Time when session was established. Is not nullable.
host_name	**nvarchar(128)**	Name of the client workstation that is specific to a session. The value is NULL for internal sessions. Is nullable. **Security Note:** The client application provides the workstation name and can provide inaccurate data. Do not rely upon HOST_NAME as a security feature.
program_name	**nvarchar(128)**	Name of client program that initiated the session. The value is NULL for internal sessions. Is nullable.
host_process_id	**int**	Process ID of the client program that initiated the session. The value is NULL for internal sessions. Is nullable.
client_version	**int**	TDS protocol version of the interface that is used by the client to connect to the server. The value is NULL for internal sessions. Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC INFORMATION
client_interface_name	**nvarchar(32)**	Name of library/driver being used by the client to communicate with the server. The value is NULL for internal sessions. Is nullable.
security_id	**varbinary(85)**	Microsoft Windows security ID associated with the login. Is not nullable.
login_name	**nvarchar(128)**	SQL Server login name under which the session is currently executing. For the original login name that created the session, see original_login_name. Can be a SQL Server authenticated login name or a Windows authenticated domain user name. Is not nullable.
nt_domain	**nvarchar(128)**	**Applies to**: SQL Server 2008 through SQL Server 2017. Windows domain for the client if the session is using Windows Authentication or a trusted connection. This value is NULL for internal sessions and non-domain users. Is nullable.
nt_user_name	**nvarchar(128)**	**Applies to**: SQL Server 2008 through SQL Server 2017. Windows user name for the client if the session is using Windows Authentication or a trusted connection. This value is NULL for internal sessions and non-domain users. Is nullable.
status	**nvarchar(30)**	Status of the session. Possible values: **Running** - Currently running one or more requests **Sleeping** - Currently running no requests **Dormant** – Session has been reset because of connection pooling and is now in prelogin state. **Preconnect** - Session is in the Resource Governor classifier. Is not nullable.
context_info	**varbinary(128)**	CONTEXT_INFO value for the session. The context information is set by the user by using the SET CONTEXT_INFO statement. Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC INFORMATION
cpu_time	**int**	CPU time, in milliseconds, that was used by this session. Is not nullable.
memory_usage	**int**	Number of 8-KB pages of memory used by this session. Is not nullable.
total_scheduled_time	**int**	Total time, in milliseconds, for which the session (requests within) were scheduled for execution. Is not nullable.
total_elapsed_time	**int**	Time, in milliseconds, since the session was established. Is not nullable.
endpoint_id	**int**	ID of the Endpoint associated with the session. Is not nullable.
last_request_start_time	**datetime**	Time at which the last request on the session began. This includes the currently executing request. Is not nullable.
last_request_end_time	**datetime**	Time of the last completion of a request on the session. Is nullable.
reads	**bigint**	Number of reads performed, by requests in this session, during this session. Is not nullable.
writes	**bigint**	Number of writes performed, by requests in this session, during this session. Is not nullable.
logical_reads	**bigint**	Number of logical reads that have been performed on the session. Is not nullable.
is_user_process	**bit**	0 if the session is a system session. Otherwise, it is 1. Is not nullable.
text_size	**int**	TEXTSIZE setting for the session. Is not nullable.
language	**nvarchar(128)**	LANGUAGE setting for the session. Is nullable.
date_format	**nvarchar(3)**	DATEFORMAT setting for the session. Is nullable.
date_first	**smallint**	DATEFIRST setting for the session. Is not nullable.
quoted_identifier	**bit**	QUOTED_IDENTIFIER setting for the session. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC INFORMATION
arithabort	**bit**	ARITHABORT setting for the session. Is not nullable.
ansi_null_dflt_on	**bit**	ANSI_NULL_DFLT_ON setting for the session. Is not nullable.
ansi_defaults	**bit**	ANSI_DEFAULTS setting for the session. Is not nullable.
ansi_warnings	**bit**	ANSI_WARNINGS setting for the session. Is not nullable.
ansi_padding	**bit**	ANSI_PADDING setting for the session. Is not nullable.
ansi_nulls	**bit**	ANSI_NULLS setting for the session. Is not nullable.
concat_null_yields_null	**bit**	CONCAT_NULL_YIELDS_NULL setting for the session. Is not nullable.
transaction_isolation_level	**smallint**	Transaction isolation level of the session. 0 = Unspecified 1 = ReadUncomitted 2 = ReadCommitted 3 = Repeatable 4 = Serializable 5 = Snapshot Is not nullable.
lock_timeout	**int**	LOCK_TIMEOUT setting for the session. The value is in milliseconds. Is not nullable.
deadlock_priority	**int**	DEADLOCK_PRIORITY setting for the session. Is not nullable.
row_count	**bigint**	Number of rows returned on the session up to this point. Is not nullable.
prev_error	**int**	ID of the last error returned on the session. Is not nullable.
original_security_id	**varbinary(85)**	Microsoft Windows security ID that is associated with the original_login_name. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC INFORMATION
original_login_name	**nvarchar(128)**	SQL Server login name that the client used to create this session. Can be a SQL Server authenticated login name, a Windows authenticated domain user name, or a contained database user. Note that the session could have gone through many implicit or explicit context switches after the initial connection. For example, if EXECUTE AS is used. Is not nullable.
last_successful_logon	**datetime**	**Applies to**: SQL Server 2008 through SQL Server 2017. Time of the last successful logon for the original_login_name before the current session started.
last_unsuccessful_logon	**datetime**	**Applies to**: SQL Server 2008 through SQL Server 2017. Time of the last unsuccessful logon attempt for the original_login_name before the current session started.
unsuccessful_logons	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2017. Number of unsuccessful logon attempts for the original_login_name between the last_successful_logon and login_time.
group_id	**int**	ID of the workload group to which this session belongs. Is not nullable.
database_id	**smallint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. ID of the current database for each session.
authenticating_database_id	**int**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. ID of the database authenticating the principal. For Logins, the value will be 0. For contained database users, the value will be the database ID of the contained database.
open_transaction_count	**int**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Number of open transactions per session.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC INFORMATION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

Everyone can see their own session information.
SQL Server: Requires `VIEW SERVER STATE` permission on SQL Server to see all sessions on the server.
SQL Database: Requires `VIEW DATABASE STATE` to see all connections to the current database.
`VIEW DATABASE STATE` cannot be granted in the `master` database.

Remarks

When the **common criteria compliance enabled** server configuration option is enabled, logon statistics are displayed in the following columns.

- last_successful_logon
- last_unsuccessful_logon
- unsuccessful_logons

  If this option is not enabled, these columns will return null values. For more information about how to set this server configuration option, see common criteria compliance enabled Server Configuration Option.

The admin connections on Azure SQL Database will see one row per authenticated session. The "sa" sessions that appear in the resultset, do not have any impact on the user quota for sessions. The non-admin connections will only see information related to their database user sessions.

Relationship Cardinalities

FROM	TO	ON/APPLY	RELATIONSHIP
sys.dm_exec_sessions	sys.dm_exec_requests	session_id	One-to-zero or one-to-many
sys.dm_exec_sessions	sys.dm_exec_connections	session_id	One-to-zero or one-to-many
sys.dm_exec_sessions	sys.dm_tran_session_transactions	session_id	One-to-zero or one-to-many
sys.dm_exec_sessions	sys.dm_exec_cursors(session_id | 0)	session_id CROSS APPLY OUTER APPLY	One-to-zero or one-to-many
sys.dm_exec_sessions	sys.dm_db_session_space_usage	session_id	One-to-one

Examples

A. Finding users that are connected to the server

The following example finds the users that are connected to the server and returns the number of sessions for each user.

```
SELECT login_name ,COUNT(session_id) AS session_count
FROM sys.dm_exec_sessions
GROUP BY login_name;
```

B. Finding long-running cursors

The following example finds the cursors that have been open for more than a specific period of time, who created the cursors, and what session the cursors are on.

```
USE master;
GO
SELECT creation_time ,cursor_id
    ,name ,c.session_id ,login_name
FROM sys.dm_exec_cursors(0) AS c
JOIN sys.dm_exec_sessions AS s
   ON c.session_id = s.session_id
WHERE DATEDIFF(mi, c.creation_time, GETDATE()) > 5;
```

C. Finding idle sessions that have open transactions

The following example finds sessions that have open transactions and are idle. An idle session is one that has no request currently running.

```
SELECT s.*
FROM sys.dm_exec_sessions AS s
WHERE EXISTS
    (
    SELECT *
    FROM sys.dm_tran_session_transactions AS t
    WHERE t.session_id = s.session_id
    )
    AND NOT EXISTS
    (
    SELECT *
    FROM sys.dm_exec_requests AS r
    WHERE r.session_id = s.session_id
    );
```

D. Finding information about a queries own connection

Typical query to gather information about a queries own connection.

```
SELECT
    c.session_id, c.net_transport, c.encrypt_option,
    c.auth_scheme, s.host_name, s.program_name,
    s.client_interface_name, s.login_name, s.nt_domain,
    s.nt_user_name, s.original_login_name, c.connect_time,
    s.login_time
FROM sys.dm_exec_connections AS c
JOIN sys.dm_exec_sessions AS s
    ON c.session_id = s.session_id
WHERE c.session_id = @@SPID;
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_exec_sql_text (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the text of the SQL batch that is identified by the specified *sql_handle*. This table-valued function replaces the system function **fn_get_sql**.

Syntax

```
sys.dm_exec_sql_text(sql_handle | plan_handle)
```

Arguments

sql_handle
Is the SQL handle of the batch to be looked up. *sql_handle* is **varbinary(64)**. *sql_handle* can be obtained from the following dynamic management objects:

- sys.dm_exec_query_stats
- sys.dm_exec_requests
- sys.dm_exec_cursors
- sys.dm_exec_xml_handles
- sys.dm_exec_query_memory_grants
- sys.dm_exec_connections

plan_handle
Uniquely identifies a query plan for a batch that is cached or is currently executing. *plan_handle* is **varbinary(64)**. *plan_handle* can be obtained from the following dynamic management objects:

- sys.dm_exec_cached_plans
- sys.dm_exec_query_stats
- sys.dm_exec_requests

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
dbid	**smallint**	ID of database. For ad hoc and prepared SQL statements, the ID of the database where the statements were compiled.

COLUMN NAME	DATA TYPE	DESCRIPTION
objectid	**int**	ID of object. Is NULL for ad hoc and prepared SQL statements.
number	**smallint**	For a numbered stored procedure, this column returns the number of the stored procedure. For more information, see sys.numbered_procedures (Transact-SQL). Is NULL for ad hoc and prepared SQL statements.
encrypted	**bit**	1 = SQL text is encrypted. 0 = SQL text is not encrypted.
text	**nvarchar(max)**	Text of the SQL query. Is NULL for encrypted objects.

Permissions

Requires `VIEW SERVER STATE` permission on the server.

Remarks

For ad hoc queries, the SQL handles are hash values based on the SQL text being submitted to the server, and can originate from any database.

For database objects such as stored procedures, triggers or functions, the SQL handles are derived from the database ID, object ID, and object number.

Plan handle is a hash value derived from the compiled plan of the entire batch.

> **NOTE**
>
> **dbid** cannot be determined from *sql_handle* for ad hoc queries. To determine **dbid** for ad hoc queries, use *plan_handle* instead.

Examples

A. Conceptual Example

The following is a basic example to illustrate passing a **sql_handle** either directly or with **CROSS APPLY**.

1. Create activity.
   Execute the following T-SQL in a new query window in SQL Server Management Studio.

```
-- Identify current spid (session_id)
SELECT @@SPID;
GO

-- Create activity
  WAITFOR DELAY '00:02:00';
```

a. Using **CROSS APPLY**.
   The sql_handle from **sys.dm_exec_requests** will be passed to **sys.dm_exec_sql_text** using **CROSS APPLY**. Open a new query window and pass the spid identified in step 1. In this example the spid happens to be `59`.

   ```
   SELECT t.*
   FROM sys.dm_exec_requests AS r
   CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) AS t
   WHERE session_id = 59 -- modify this value with your actual spid
   ```

b. Passing **sql_handle** directly.
   Acquire the **sql_handle** from **sys.dm_exec_requests**. Then, pass the **sql_handle** directly to **sys.dm_exec_sql_text**. Open a new query window and pass the spid identified in step 1 to **sys.dm_exec_requests**. In this example the spid happens to be `59`. Then pass the returned **sql_handle** as an argument to **sys.dm_exec_sql_text**.

   ```
   -- acquire sql_handle
   SELECT sql_handle FROM sys.dm_exec_requests WHERE session_id = 59  -- modify this value with
   your actual spid

   -- pass sql_handle to sys.dm_exec_sql_text
   SELECT * FROM
   sys.dm_exec_sql_text(0x01000600B74C2A1300D2582A2100000000000000000000000000000000000000000
   0000000000000000) -- modify this value with your actual sql_handle
   ```

B. Obtain information about the top five queries by average CPU time

The following example returns the text of the SQL statement and average CPU time for the top five queries.

```
SELECT TOP 5 total_worker_time/execution_count AS [Avg CPU Time],
    SUBSTRING(st.text, (qs.statement_start_offset/2)+1,
        ((CASE qs.statement_end_offset
          WHEN -1 THEN DATALENGTH(st.text)
         ELSE qs.statement_end_offset
         END - qs.statement_start_offset)/2) + 1) AS statement_text
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS st
ORDER BY total_worker_time/execution_count DESC;
```

C. Provide batch-execution statistics

The following example returns the text of SQL queries that are being executed in batches and provides statistical information about them.

```
SELECT s2.dbid,
    s1.sql_handle,
    (SELECT TOP 1 SUBSTRING(s2.text,statement_start_offset / 2+1 ,
      ( (CASE WHEN statement_end_offset = -1
         THEN (LEN(CONVERT(nvarchar(max),s2.text)) * 2)
         ELSE statement_end_offset END)  - statement_start_offset) / 2+1))  AS sql_statement,
    execution_count,
    plan_generation_num,
    last_execution_time,
    total_worker_time,
    last_worker_time,
    min_worker_time,
    max_worker_time,
    total_physical_reads,
    last_physical_reads,
    min_physical_reads,
    max_physical_reads,
    total_logical_writes,
    last_logical_writes,
    min_logical_writes,
    max_logical_writes
FROM sys.dm_exec_query_stats AS s1
CROSS APPLY sys.dm_exec_sql_text(sql_handle) AS s2
WHERE s2.objectid is null
ORDER BY s1.sql_handle, s1.statement_start_offset, s1.statement_end_offset;
```

See also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_query_stats (Transact-SQL)
sys.dm_exec_requests (Transact-SQL)
sys.dm_exec_cursors (Transact-SQL)
sys.dm_exec_xml_handles (Transact-SQL)
sys.dm_exec_query_memory_grants (Transact-SQL)
Using APPLY
sys.dm_exec_text_query_plan (Transact-SQL)

sys.dm_exec_text_query_plan (Transact-SQL)

5/4/2018 • 6 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the Showplan in text format for a Transact-SQL batch or for a specific statement within the batch. The query plan specified by the plan handle can either be cached or currently executing. This table-valued function is similar to sys.dm_exec_query_plan (Transact-SQL), but has the following differences:

- The output of the query plan is returned in text format.
- The output of the query plan is not limited in size.
- Individual statements within the batch can be specified.

Applies to: SQL Server (SQL Server 2008 through current version), Azure SQL Database.

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_exec_text_query_plan
(
    plan_handle
    , { statement_start_offset | 0 | DEFAULT }
        , { statement_end_offset | -1 | DEFAULT }
)
```

Arguments

plan_handle
Uniquely identifies a query plan for a batch that is cached or is currently executing. *plan_handle* is **varbinary(64)**.

The plan handle can be obtained from the following dynamic management objects:

- sys.dm_exec_cached_plans
- sys.dm_exec_query_stats
- sys.dm_exec_requests

statement_start_offset | 0 | DEFAULT
Indicates, in bytes, the starting position of the query that the row describes within the text of its batch or persisted object. *statement_start_offset* is **int**. A value of 0 indicates the beginning of the batch. The default value is 0.

The statement start offset can be obtained from the following dynamic management objects:

- sys.dm_exec_query_stats
- sys.dm_exec_requests

statement_end_offset | -1 | DEFAULT
Indicates, in bytes, the ending position of the query that the row describes within the text of its batch or persisted object.

statement_start_offset is **int**.

A value of -1 indicates the end of the batch. The default value is -1.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
dbid	**smallint**	ID of the context database that was in effect when the Transact-SQL statement corresponding to this plan was compiled. For ad hoc and prepared SQL statements, the ID of the database where the statements were compiled. Column is nullable.
objectid	**int**	ID of the object (for example, stored procedure or user-defined function) for this query plan. For ad hoc and prepared batches, this column is **null**. Column is nullable.
number	**smallint**	Numbered stored procedure integer. For example, a group of procedures for the **orders** application may be named **orderproc;1**, **orderproc;2**, and so on. For ad hoc and prepared batches, this column is **null**. Column is nullable.
encrypted	**bit**	Indicates whether the corresponding stored procedure is encrypted. 0 = not encrypted 1 = encrypted Column is not nullable.
query_plan	**nvarchar(max)**	Contains the compile-time Showplan representation of the query execution plan that is specified with *plan_handle*. The Showplan is in text format. One plan is generated for each batch that contains, for example ad hoc Transact-SQL statements, stored procedure calls, and user-defined function calls. Column is nullable.

Remarks

Under the following conditions, no Showplan output is returned in the **plan** column of the returned table for **sys.dm_exec_text_query_plan**:

- If the query plan that is specified by using *plan_handle* has been evicted from the plan cache, the **query_plan** column of the returned table is null. For example, this condition may occur if there is a time delay between when the plan handle was captured and when it was used with **sys.dm_exec_text_query_plan**.

- Some Transact-SQL statements are not cached, such as bulk operation statements or statements containing string literals larger than 8 KB in size. Showplans for such statements cannot be retrieved by using **sys.dm_exec_text_query_plan** because they do not exist in the cache.
- If a Transact-SQL batch or stored procedure contains a call to a user-defined function or a call to dynamic SQL, for example using EXEC (*string*), the compiled XML Showplan for the user-defined function is not included in the table returned by **sys.dm_exec_text_query_plan** for the batch or stored procedure. Instead, you must make a separate call to **sys.dm_exec_text_query_plan** for the *plan_handle* that corresponds to the user-defined function.

When an ad hoc query uses simple or forced parameterization, the **query_plan** column will contain only the statement text and not the actual query plan. To return the query plan, call **sys.dm_exec_text_query_plan** for the plan handle of the prepared parameterized query. You can determine whether the query was parameterized by referencing the **sql** column of the sys.syscacheobjects view or the text column of the sys.dm_exec_sql_text dynamic management view.

Permissions

To execute **sys.dm_exec_text_query_plan**, a user must be a member of the **sysadmin** fixed server role or have the VIEW SERVER STATE permission on the server.

Examples

A. Retrieving the cached query plan for a slow-running Transact-SQL query or batch

If a Transact-SQL query or batch runs a long time on a particular connection to SQL Server, retrieve the execution plan for that query or batch to discover what is causing the delay. The following example shows how to retrieve the Showplan for a slow-running query or batch.

> **NOTE**
>
> To run this example, replace the values for *session_id* and *plan_handle* with values specific to your server.

First, retrieve the server process ID (SPID) for the process that is executing the query or batch by using the `sp_who` stored procedure:

```
USE master;
GO
EXEC sp_who;
GO
```

The result set that is returned by `sp_who` indicates that the SPID is `54`. You can use the SPID with the `sys.dm_exec_requests` dynamic management view to retrieve the plan handle by using the following query:

```
USE master;
GO
SELECT * FROM sys.dm_exec_requests
WHERE session_id = 54;
GO
```

The table that is returned by **sys.dm_exec_requests** indicates that the plan handle for the slow-running query or batch is `0x06000100A27E7C1FA821B10600`. The following example returns the query plan for the specified plan handle and uses the default values 0 and -1 to return all statements in the query or batch.

```
USE master;
GO
SELECT query_plan
FROM sys.dm_exec_text_query_plan (0x06000100A27E7C1FA821B10600,0,-1);
GO
```

B. Retrieving every query plan from the plan cache

To retrieve a snapshot of all query plans residing in the plan cache, retrieve the plan handles of all query plans in the cache by querying the `sys.dm_exec_cached_plans` dynamic management view. The plan handles are stored in the `plan_handle` column of `sys.dm_exec_cached_plans`. Then use the CROSS APPLY operator to pass the plan handles to `sys.dm_exec_text_query_plan` as follows. The Showplan output for each plan currently in the plan cache is in the `query_plan` column of the table that is returned.

```
USE master;
GO
SELECT *
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_text_query_plan(cp.plan_handle, DEFAULT, DEFAULT);
GO
```

C. Retrieving every query plan for which the server has gathered query statistics from the plan cache

To retrieve a snapshot of all query plans for which the server has gathered statistics that currently reside in the plan cache, retrieve the plan handles of these plans in the cache by querying the `sys.dm_exec_query_stats` dynamic management view. The plan handles are stored in the `plan_handle` column of `sys.dm_exec_query_stats`. Then use the CROSS APPLY operator to pass the plan handles to `sys.dm_exec_text_query_plan` as follows. The Showplan output for each plan is in the `query_plan` column of the table that is returned.

```
USE master;
GO
SELECT * FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_text_query_plan(qs.plan_handle, qs.statement_start_offset, qs.statement_end_offset);
GO
```

D. Retrieving information about the top five queries by average CPU time

The following example returns the query plans and average CPU time for the top five queries. The **sys.dm_exec_text_query_plan** function specifies the default values 0 and -1 to return all statements in the batch in the query plan.

```
SELECT TOP 5 total_worker_time/execution_count AS [Avg CPU Time],
Plan_handle, query_plan
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_text_query_plan(qs.plan_handle, 0, -1)
ORDER BY total_worker_time/execution_count DESC;
GO
```

See Also

sys.dm_exec_query_plan (Transact-SQL)

sys.dm_exec_trigger_stats (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns aggregate performance statistics for cached triggers. The view contains one row per trigger, and the lifetime of the row is as long as the trigger remains cached. When a trigger is removed from the cache, the corresponding row is eliminated from this view. At that time, a Performance Statistics SQL trace event is raised similar to **sys.dm_exec_query_stats**.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	Database ID in which the trigger resides.
object_id	**int**	Object identification number of the trigger.
type	**char(2)**	Type of the object: TA = Assembly (CLR) trigger TR = SQL trigger
Type_desc	**nvarchar(60)**	Description of the object type: CLR_TRIGGER SQL_TRIGGER
sql_handle	**varbinary(64)**	This can be used to correlate with queries in **sys.dm_exec_query_stats** that were executed from within this trigger.
plan_handle	**varbinary(64)**	Identifier for the in-memory plan. This identifier is transient and remains constant only while the plan remains in the cache. This value may be used with the **sys.dm_exec_cached_plans** dynamic management view.
cached_time	**datetime**	Time at which the trigger was added to the cache.
last_execution_time	**datetime**	Last time at which the trigger was executed.
execution_count	**bigint**	The number of times that the trigger has been executed since it was last compiled.

COLUMN NAME	DATA TYPE	DESCRIPTION
total_worker_time	**bigint**	The total amount of CPU time, in microseconds, that was consumed by executions of this trigger since it was compiled.
last_worker_time	**bigint**	CPU time, in microseconds, that was consumed the last time the trigger was executed.
min_worker_time	**bigint**	The maximum CPU time, in microseconds, that this trigger has ever consumed during a single execution.
max_worker_time	**bigint**	The maximum CPU time, in microseconds, that this trigger has ever consumed during a single execution.
total_physical_reads	**bigint**	The total number of physical reads performed by executions of this trigger since it was compiled.
last_physical_reads	**bigint**	The number of physical reads performed the last time the trigger was executed.
min_physical_reads	**bigint**	The minimum number of physical reads that this trigger has ever performed during a single execution.
max_physical_reads	**bigint**	The maximum number of physical reads that this trigger has ever performed during a single execution.
total_logical_writes	**bigint**	The total number of logical writes performed by executions of this trigger since it was compiled.
last_logical_writes	**bigint**	The number of logical writes performed the last time the trigger was executed.
min_logical_writes	**bigint**	The minimum number of logical writes that this trigger has ever performed during a single execution.
max_logical_writes	**bigint**	The maximum number of logical writes that this trigger has ever performed during a single execution.
total_logical_reads	**bigint**	The total number of logical reads performed by executions of this trigger since it was compiled.
last_logical_reads	**bigint**	The number of logical reads performed the last time the trigger was executed.

COLUMN NAME	DATA TYPE	DESCRIPTION
min_logical_reads	**bigint**	The minimum number of logical reads that this trigger has ever performed during a single execution.
max_logical_reads	**bigint**	The maximum number of logical reads that this trigger has ever performed during a single execution.
total_elapsed_time	**bigint**	The total elapsed time, in microseconds, for completed executions of this trigger.
last_elapsed_time	**bigint**	Elapsed time, in microseconds, for the most recently completed execution of this trigger.
min_elapsed_time	**bigint**	The minimum elapsed time, in microseconds, for any completed execution of this trigger.
max_elapsed_time	**bigint**	The maximum elapsed time, in microseconds, for any completed execution of this trigger.
total_spills	**bigint**	The total number of pages spilled by execution of this trigger since it was compiled. **Applies to**: Starting with SQL Server 2017 (14.x) CU3
last_spills	**bigint**	The number of pages spilled the last time the trigger was executed. **Applies to**: Starting with SQL Server 2017 (14.x) CU3
min_spills	**bigint**	The minimum number of pages that this trigger has ever spilled during a single execution. **Applies to**: Starting with SQL Server 2017 (14.x) CU3
max_spills	**bigint**	The maximum number of pages that this trigger has ever spilled during a single execution. **Applies to**: Starting with SQL Server 2017 (14.x) CU3

Remarks

In SQL Database, dynamic management views cannot expose information that would impact database containment or expose information about other databases the user has access to. To avoid exposing this information, every row that contains data that doesn't belong to the connected tenant is filtered out.

Statistics in the view are updated when a query is completed.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example returns information about the top five triggers identified by average elapsed time.

```
SELECT TOP 5 d.object_id, d.database_id, DB_NAME(database_id) AS 'database_name',
    OBJECT_NAME(object_id, database_id) AS 'trigger_name', d.cached_time,
    d.last_execution_time, d.total_elapsed_time,
    d.total_elapsed_time/d.execution_count AS [avg_elapsed_time],
    d.last_elapsed_time, d.execution_count
FROM sys.dm_exec_trigger_stats AS d
ORDER BY [total_worker_time] DESC;
```

See Also

Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_exec_sql_text (Transact-SQL)
sys.dm_exec_query_stats (Transact-SQL)
sys.dm_exec_procedure_stats (Transact-SQL)
sys.dm_exec_cached_plans (Transact-SQL)

sys.dm_exec_valid_use_hints (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns USE HINT supported hint names. It lists one hint name per row.

Use this DMV to see the list of all supported hints under the USE HINT notation.

COLUMN NAME	DATA TYPE	DESCRIPTION
name	**sysname**	The name of the hint.

See Query Hints for descriptions of each hint.

Introduced in SQL Server 2016 (13.x) SP1.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Database Related Dynamic Management Views (Transact-SQL)

sys.dm_exec_xml_handles (Transact-SQL)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about active handles that have been opened by **sp_xml_preparedocument**.

Syntax

```
dm_exec_xml_handles (session_id | 0 )
```

Arguments

session_id | 0,
ID of the session. If *session_id* is specified, this function returns information about XML handles in the specified session.

If 0 is specified, the function returns information about all XML handles for all sessions.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**int**	Session ID of the session that holds this XML document handle.
document_id	**int**	XML document handle ID returned by **sp_xml_preparedocument**.
namespace_document_id	**int**	Internal handle ID used for the associated namespace document that has been passed as the third parameter to **sp_xml_preparedocument**. NULL if there is no namespace document.
sql_handle	**varbinary(64)**	Handle to the text of the SQL code where the handle has been defined.
statement_start_offset	**int**	Number of characters into the currently executing batch or stored procedure at which the **sp_xml_preparedocument** call occurs. Can be used together with the **sql_handle**, the **statement_end_offset**, and the **sys.dm_exec_sql_text** dynamic management function to retrieve the currently executing statement for the request.

COLUMN NAME	DATA TYPE	DESCRIPTION
statement_end_offset	**int**	Number of characters into the currently executing batch or stored procedure at which the **sp_xml_preparedocument** call occurs. Can be used together with the **sql_handle**, the **statement_start_offset**, and the **sys.dm_exec_sql_text** dynamic management function to retrieve the currently executing statement for the request.
creation_time	**datetime**	Timestamp when **sp_xml_preparedocument** was called.
original_document_size_bytes	**bigint**	Size of the unparsed XML document in bytes.
original_namespace_document_size_bytes	**bigint**	Size of the unparsed XML namespace document, in bytes. NULL if there is no namespace document.
num_openxml_calls	**bigint**	Number of OPENXML calls with this document handle.
row_count	**bigint**	Number of rows returned by all previous OPENXML calls for this document handle.
dormant_duration_ms	**bigint**	Milliseconds since the last OPENXML call. If OPENXML has not been called, returns milliseconds since the **sp_xml_preparedocumen**t call.

Remarks

The lifetime of **sql_handles** used to retrieve the SQL text that executed a call to **sp_xml_preparedocument** outlives the cached plan used to execute the query. If the query text is not available in the cache, the data cannot be retrieved using the information provided in the function result. This can occur if you are running many large batches.

Permissions

Requires VIEW SERVER STATE permission on the server to see all sessions or session IDs that are not owned by the caller. A caller can always see the data for his or her own current session ID.

Examples

The following example selects all active handles.

```
SELECT * FROM sys.dm_exec_xml_handles(0);
```

See Also

[Dynamic Management Views and Functions (Transact-SQL)]

[Execution Related Dynamic Management Views and Functions (Transact-SQL)]

[sp_xml_preparedocument (Transact-SQL)]

[sp_xml_removedocument (Transact-SQL)]

sys.dm_external_script_execution_stats

5/3/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row for each type of external script request. The external script requests are grouped by the supported external script language. One row is generated for each registered external script functions. Arbitrary external script functions are not recorded unless sent by a parent process, such as `rxExec`.

> **NOTE**
> This DMV is available only if you have installed and then enabled the feature that supports external script execution. For information about how to do this for R scripts, see Set up SQL Server R services.

COLUMN NAME	DATA TYPE	DESCRIPTION
language	**nvarchar**	Name of the registered external script language. Each external script must specify the language in the script request to start the associated launcher.
counter_name	**nvarchar**	Name of a registered external script function. Is not nullable.
counter_value	**integer**	Total number of instances that the registered external script function has been called on the server. This value is cumulative, beginning with the time that the feature was installed on the instance, and cannot be reset.

Permissions

Requires VIEW SERVER STATE permission on server.

> **NOTE**
> Users who run external scripts must have the additional permission EXECUTE ANY EXTERNAL SCRIPT, however, this DMV can be used by administrators without this permission.

Remarks

This DMV is provided for internal telemetry, to monitor overall usage of the new external script execution feature provided in SQL Server. The telemetry service starts when LaunchPad does and increments a disk-based counter each time a registered external script function is called.

Generally speaking, performance counters are valid only as long as the process that generated them is active. Therefore, a query on a DMV cannot show detailed data for services that have stopped running. For example, if a launcher executes external script and yet completes them very quickly, a convnetional DMV might not show any data

Therefore, the counters tracked by this DMV are kept running, and state for sys.dm_external_script_requests is preserved by using writes to disk, even if the instance is shut down.

R Counter Values

Currently the only external script language supported in SQL Server 2017 is R. External script requests for the R language are handled by R Services (In-Database).

For R, this DMV tracks the number of R calls that are made on an instance. For example, if `rxLinMod` is called and run in parallel, the counter is incremented by 1.

For the R language, the counter values displayed in the *counter_name* field represent the names of registered ScaleR functions. The values in the *counter_value* field represent the cumulative number of instances that the specific ScaleR function.

The count begins when the feature is installed and enabled on the instance and is cumulative until the file that maintains state is deleted or overwritten by an administrator. Therefore, it is generally not possible to reset the values in *counter_value*. If you want to monitor usage by session, calendar times, or other intervals, we recommend that you capture the counts to a table.

Registration of external script functions

The R language supports arbitrary scripts, and the R community provides many thousand packages, each with their own functions and methods. However, this DMV monitors only the ScaleR functions that are installed with SQL Server R Services.

Registration of these functions is performed when the feature is installed, and registered functions cannot be added or deleted.

Examples

Viewing the number of R scripts run on the server

The following example displays the cumulative number of external script executions for the R language.

```
SELECT counter_name, counter_value
FROM sys.dm_external_script_execution_stats
WHERE language = 'R';
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL) sys.dm_external_script_requests
sp_execute_external_script

sys.dm_external_script_requests

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each active worker account that is running an external script.

> **NOTE**
> This DMV is available only if you have installed and then enabled the feature that supports external script execution. For information about how to do this for R scripts, see Set Up SQL Server R Services.

COLUMN NAME	DATA TYPE	DESCRIPTION
external_script_request_id	**unique identifier**	ID of the process that sent the external script request. This corresponds to the process ID as received by SQL Server
language	**nvarchar**	Keyword that represents a supported script language. Currently only `R` is supported.
degree_of_parallelism	**int**	Number indicating the number of parallel processes that were created. This value might be different from the number of parallel processes that were requested.
external_user_name	**nvarchar**	The Windows worker account under which the script was executed.

Permissions

Requires VIEW SERVER STATE permission on server.

> **NOTE**
> Users who run external scripts must have the additional permission EXECUTE ANY EXTERNAL SCRIPT, however, this DMV can be used by administrators without this permission.

Remarks

This view can be filtered using the script language identifier.

The view also returns the worker account under which the script is being run. For information about worker accounts used by R scripts, see Modify the User Account Pool for R Services.

The GUID that is returned in the **external_script_request_id** field also represents the file name of the secured directory where temporary files are stored. Each worker account such as MSSQLSERVER01 represents a single SQL login or Windows user, and might be used to run multiple script requests. By default, these temporary files are cleaned up after completion of the requested script. If you need to preserve these files for some period for debugging purposes, you can change the cleanup flag as described in this topic: Configure and Manage Advanced Analytics Extensions.

This DMV monitors only active processes and cannot report on scripts that have already completed. If you need to track the duration of scripts, we recommend that you add timing information into your script and capture that as part of script execution.

Examples

Viewing the currently active R scripts for a particular process

The following example displays the number of external script executions being run on the current instance.

```
SELECT external_script_request_id
  , [language]
  , degree_of_parallelism
  , external_user_name
FROM sys.dm_external_script_requests;
```

Results

EXTERNAL_SCRIPT_REQUEST_ID	LANGUAGE	DEGREE_OF_PARALLELISM	EXTERNAL_USER_NAME
183EE6FC-7399-4318-AA2E-7A6C68E435A8	R	1	MSSQLSERVER01

See Also

Dynamic Management Views and Functions (Transact-SQL)
Execution Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_external_script_execution_stats sp_execute_external_script

Extended Events Dynamic Management Views

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management objects for SQL Server Extended Events.

sys.dm_xe_map_values (Transact-SQL)	sys.dm_xe_session_events (Transact-SQL)
sys.dm_xe_object_columns (Transact-SQL)	sys.dm_xe_session_object_columns (Transact-SQL)
sys.dm_xe_objects (Transact-SQL)	sys.dm_xe_session_targets (Transact-SQL)
sys.dm_xe_packages (Transact-SQL)	sys.dm_xe_sessions (Transact-SQL)
sys.dm_xe_session_event_actions (Transact-SQL)	

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_xe_database_sessions (Azure SQL Database)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about session events. Events are discrete execution points. Predicates can be applied to events to stop them from firing if the event does not contain the required information.

Applies to: Azure SQL Database V12 and any later versions.

COLUMN NAME	DATA TYPE	DESCRIPTION
event_session_address	**varbinary(8)**	The memory address of the event session. Is not nullable.
event_name	**nvarchar(60)**	The name of the event that an action is bound to. Is not nullable.
event_package_guid	**uniqueidentifier**	The GUID for the package containing the event. Is not nullable.
event_predicate	**nvarchar(2048)**	An XML representation of the predicate tree that is applied to the event. Is nullable.

Permissions

Requires VIEW DATABASE STATE permission.

Relationship Cardinalities

As of 2015-07-13, 'sys.dm_xe_objects' is one of these XEvents DMVs that do Not contain '_database' in their name. Not a typo or error in the following table's right-side column. The name is the same in Microsoft SQL Server and Azure SQL Database. GeneMi.

FROM	TO	RELATIONSHIP
sys.dm_xe_database_session_events.event_session_address	sys.dm_xe_database_sessions.address	Many-to-one
sys.dm_xe_database_session_events.event_package_guid, sys.dm_xe_database_session_events.event_name	sys.dm_xe_objects.name, sys.dm_xe_objects.package_guid	Many-to-one

See Also

Extended events in Azure SQL Database
Extended Events

sys.dm_xe_database_session_targets (Azure SQL Database)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about session targets.

Applies to: Azure SQL Database V12 and any future versions.

COLUMN NAME	DATA TYPE	DESCRIPTION
event_session_address	**varbinary(8)**	The memory address of the event session. Has a many-to-one relationship with sys.dm_xe_database_sessions.address. Is not nullable.
target_name	**nvarchar(60)**	The name of the target within a session. Is not nullable.
target_package_guid	**uniqueidentifier**	The GUID of the package that contains the target. Is not nullable.
execution_count	**bigint**	The number of times the target has been executed for the session. Is not nullable.
execution_duration_ms	**bigint**	The total amount of time, in milliseconds, that the target has been executing. Is not nullable.
target_data	**nvarchar(max)**	The data that the target maintains, such as event aggregation information. Is nullable.

Permissions

Requires VIEW DATABASE STATE permission.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_xe_database_session_targets.event_session_address	sys.dm_xe_database_sessions.address	Many-to-one

sys.dm_xe_database_session_object_columns (Azure SQL Database)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Shows the configuration values for objects that are bound to a session.

Applies to: Azure SQL Database V12 and any later versions.

COLUMN NAME	DATA TYPE	DESCRIPTION
event_session_address	**varbinary(8)**	The memory address of the event session. Has a many-to-one relationship with sys.dm_xe_database_sessions.address. Is not nullable.
column_name	**nvarchar(60)**	The name of the configuration value. Is not nullable.
column_id	**int**	The ID of the column. Is unique within the object. Is not nullable.
column_value	**nvarchar(2048)**	The configured value of the column. Is nullable.
object_type	**nvarchar(60)**	The type of the object. Is not nullable.object_type is one of: event target
object_name	**nvarchar(60)**	The name of the object to which this column belongs. Is not nullable.
object_package_guid	**uniqueidentifier**	The GUID of the package that contains the object. Is not nullable.

Permissions

Requires VIEW DATABASE STATE permission.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_xe_database_session_object_columns.object_name dm_xe_database_session_object_columns.object_package_guid	sys.dm_xe_objects.package_guid sys.dm_xe_objects.name	Many-to-one
dm_xe_database_session_object_columns.column_name dm_xe_database_session_object_columns.column_id	sys.dm_xe_object_columns.name sys.dm_xe_object_columns.column_id	Many-to-one

See Also

Extended Events

sys.dm_xe_database_session_events (Azure SQL Database)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about session events. Events are discrete execution points. Predicates can be applied to events to stop them from firing if the event does not contain the required information.

Applies to: Azure SQL Database V12 and any later versions.

COLUMN NAME	DATA TYPE	DESCRIPTION
event_session_address	**varbinary(8)**	The memory address of the event session. Is not nullable.
event_name	**nvarchar(60)**	The name of the event that an action is bound to. Is not nullable.
event_package_guid	**uniqueidentifier**	The GUID for the package containing the event. Is not nullable.
event_predicate	**nvarchar(2048)**	An XML representation of the predicate tree that is applied to the event. Is nullable.

Permissions

Requires VIEW DATABASE STATE permission.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_xe_database_session_events.event_session_address	sys.dm_xe_database_sessions.address	Many-to-one
sys.dm_xe_database_session_events.event_package_guid, sys.dm_xe_database_session_events.event_name	sys.dm_xe_objects.name, sys.dm_xe_objects.package_guid	Many-to-one

sys.dm_xe_database_session_event_actions (Azure SQL Database)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about event session actions. Actions are executed when events are fired. This management view aggregates statistics about the number of times an action has run, and the total run time of the action.

Applies to: Azure SQL Database V12 and any future versions.

COLUMN NAME	DATA TYPE	DESCRIPTION
event_session_address	**varbinary(8)**	The memory address of the event session. Is not nullable.
action_name	**nvarchar(60)**	The name of the action. Is not nullable.
action_package_guid	**uniqueidentifier**	The GUID for the package that contains the action. Is not nullable.
event_name	**nvarchar(60)**	The name of the event that the action is bound to. Is not nullable.
event_package_guid	**uniqueidentifier**	The GUID for the package that contains the event. Is not nullable.

Permissions

Requires VIEW DATABASE STATE permission.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_xe_database_session_event_actions.event_session_address	sys.dm_xe_database_sessions.address	Many-to-one
sys.dm_xe_database_session_event_actions.action_name sys.dm_xe_session_event_actions.action_package_guid	sys.dm_xe_objects.name sys.dm_xe_database_session_events.event_package_guid	Many-to-one
sys.dm_xe_database_session_event_actions.event_name sys.dm_xe_database_session_event_actions.event_package_guid	sys.dm_xe_objects.name sys.dm_xe_objects.package_guid	Many-to-one

See Also

Extended Events

sys.dm_xe_map_values (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a mapping of internal numeric keys to human-readable text.

COLUMN NAME	DATA TYPE	DESCRIPTION
name	**nvarchar(60)**	The name of the map. name is unique across the local system. Is not nullable.
object_package_guid	**uniqueidentifier**	The GUID of the package that contains the map. Is not nullable.
map_key	**int**	The internal key value. Is not nullable.
map_value	**nvarchar(2048)**	A description of the key value. Is not nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_xe_map_values.object_package_guid dm_xe_map_values.name	sys.dm_xe_objects.package_guid sys.dm_xe_objects.name	Many-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_xe_object_columns (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the schema information for all the objects.

> **NOTE**
>
> Event objects expose fixed schemas for both read-only and read-write data.

COLUMN NAME	DATA TYPE	DESCRIPTION
name	**nvarchar(60)**	The name of the column. name is unique within the object. Is not nullable.
column_id	**int**	The identifier of the column. column_id is unique within the object when used with column_type. Is not nullable.
object_name	**nvarchar(60)**	The name of the object to which this column belongs. There is a many-to-one relationship with sys.dm_xe_objects.id. Is not nullable.
object_package_guid	**uniqueidentifier**	The GUID of the package that contains the object. Is not nullable.
type_name	**nvarchar(60)**	The name of the type for this column. Is not nullable.
type_package_guid	**uniqueidentifier**	The GUID of the package that contains the column data type. Is not nullable.
column_type	**nvarchar(60)**	Indicates how this column is used. Is not nullable. column_type can be one of the following: readonly. The column contains a static value that cannot be changed. data. The column contains run-time data exposed by the object. customizable. The column contains a value that can be changed. Note: Changing this value can modify the behavior of the object.
column_value	**nvarchar(256)**	Displays static values associated with the object column. Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
capabilities	**int**	A bitmap describing the capabilities of the column. Is nullable.
capabilities_desc	**nvarchar(256)**	A description of this object column's capabilities. This value can be one of the following: Mandatory. The value must be set when binding the parent object to an event session. NULL
description	**nvarchar(256)**	The description of this object column. Is nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_xe_object_columns.object_name, sys.dm_xe_object_columns.object_package_guid	sys.dm_xe_objects.name, sys.dm_xe_objects.package_guid	Many-to-one
sys.dm_xe_object_columns.type_name sys.dm_xe_object_columns.type_package_guid	sys.dm_xe_objects.name sys.dm_xe_objects.package_guid	Many-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_xe_objects (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each object that is exposed by an event package. Objects can be one of the following:

- Events. Events indicate points of interest in an execution path. All events contain information about a point of interest.
- Actions. Actions are run synchronously when events fire. An action can append run time data to an event.
- Targets. Targets consume events, either synchronously on the thread that fires the event or asynchronously on a system-provided thread.
- Predicates. Predicate sources retrieve values from event sources for use in comparison operations. Predicate comparisons compare specific data types and return a Boolean value.
- Types. Types encapsulate the length and characteristics of the byte collection, which is required in order to interpret the data.

COLUMN NAME	DATA TYPE	DESCRIPTION
name	**nvarchar(60)**	The name of the object. name is unique within a package for a specific object type. Is not nullable.
object_type	**nvarchar(60)**	The type of the object. object_type is one of the following: event action target pred_source pred_compare type Is not nullable.
package_guid	**uniqueidentifier**	The GUID for the package that exposes this action. There is a many-to-one relationship with sys.dm_xe_packages.package_id. Is not nullable.
description	**nvarchar(256)**	A description of the action. description is set by the package author. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
capabilities	**int**	A bitmap that describes the capabilities of the object. Is nullable.
capabilities_desc	**nvarchar(256)**	Lists all the capabilities of the object. Is nullable. **Capabilities that apply to all object types** — **Private**. The only object available for internal use, and that cannot be accessed via the CREATE/ALTER EVENT SESSION DDL. Audit events and targets fall into this category in addition to a small number of objects used internally. =============== **Event Capabilities** — **No_block**. The event is in a critical code path that cannot block for any reason. Events with this capability may not be added to any event session that specifies NO_EVENT_LOSS. =============== **Capabilities that apply to all object types** — **Process_whole_buffers**. The target consumes buffers of events at a time, rather than event by event. — **Singleton**. Only one instance of the target can exist in a process. Although multiple event sessions can reference the same singleton target there is really only one instance, and that instance will see each unique event only once. This is important if the target is added to multiple sessions that all collect the same event. — **Synchronous**. The target is executed on the thread that is producing the event, before control is returned to the calling code line.
type_name	**nvarchar(60)**	The name for pred_source and pred_compare objects. Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
type_package_guid	**uniqueidentifier**	The GUID for the package that exposes the type that this object operates on. Is nullable.
type_size	**int**	The size, in bytes, of the data type. This is only for valid object types. Is nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_xe_objects.package_guid	sys.dm_xe_packages.guid	Many-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_xe_packages (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Lists all the packages registered with the extended events engine.

COLUMN NAME	DATA TYPE	DESCRIPTION
name	**nvarchar(60)**	The name of package. The description is exposed from the package itself. Is not nullable.
guid	**uniqueidentifier**	The GUID that identifies the package. Is not nullable.
description	**nvarchar(256)**	The package description. descriptionis set by the package author and is not nullable.
capabilities	**int**	Bitmap describing the capabilities of this package. Is nullable.
capabilities_desc	**nvarchar(256)**	A list of all the capabilities possible for this package. Is nullable.
module_guid	**uniqueidentifier**	The GUID of the module that exposes this package. Is not nullable.
module_address	**varbinary(8)**	The base address where the module containing the package is loaded. A single module may expose several packages. Is not nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Remarks

The packages registered with the extended events engine expose events, the actions that can be taken at the time of event firing, and targets for both synchronous and asynchronous processing of event data.

These packages can be dynamically loaded into a process address space. At the time the package is loaded, it registers all the objects it exposes with the extended events engine.

Relationship Cardinalities

From	To	Relationship

sys.dm_xe_packages.module_address	sys.dm_os_loaded_modules.base_address	Many to one

See also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_xe_session_event_actions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about event session actions. Actions are executed when events are fired. This management view aggregates statistics about the number of times an action has run, and the total run time of the action.

COLUMN NAME	DATA TYPE	DESCRIPTION
event_session_address	**varbinary(8)**	The memory address of the event session. Is not nullable.
action_name	**nvarchar(60)**	The name of the action. Is not nullable.
action_package_guid	**uniqueidentifier**	The GUID for the package that contains the action. Is not nullable.
event_name	**nvarchar(60)**	The name of the event that the action is bound to. Is not nullable.
event_package_guid	**uniqueidentifier**	The GUID for the package that contains the event. Is not nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_xe_session_event_actions.event_session_address	sys.dm_xe_sessions.address	Many-to-one
sys.dm_xe_session_event_actions.action_name sys.dm_xe_session_event_actions.action_package_guid	sys.dm_xe_objects.name sys.dm_xe_session_events.event_package_guid	Many-to-one
sys.dm_xe_session_event_actions.event_name sys.dm_xe_session_event_actions.event_package_guid	sys.dm_xe_objects.name sys.dm_xe_objects.package_guid	Many-to-one

Change History

UPDATED CONTENT
Updated "Relationship Cardinalities" table with the correct dynamic management view names and column names.

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_xe_session_events (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about session events. Events are discrete execution points. Predicates can be applied to events to stop them from firing if the event does not contain the required information.

COLUMN NAME	DATA TYPE	DESCRIPTION
event_session_address	**varbinary(8)**	The memory address of the event session. Is not nullable.
event_name	**nvarchar(60)**	The name of the event that an action is bound to. Is not nullable.
event_package_guid	**uniqueidentifier**	The GUID for the package containing the event. Is not nullable.
event_predicate	**nvarchar(2048)**	An XML representation of the predicate tree that is applied to the event. Is nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_xe_session_events.event_session_address	sys.dm_xe_sessions.address	Many-to-one
sys.dm_xe_session_events.event_package_guid, sys.dm_xe_session_events.event_name	sys.dm_xe_objects.name, sys.dm_xe_objects.package_guid	Many-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_xe_session_object_columns (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Shows the configuration values for objects that are bound to a session.

COLUMN NAME	DATA TYPE	DESCRIPTION
event_session_address	**varbinary(8)**	The memory address of the event session. Has a many-to-one relationship with sys.dm_xe_sessions.address. Is not nullable.
column_name	**nvarchar(60)**	The name of the configuration value. Is not nullable.
column_id	**int**	The ID of the column. Is unique within the object. Is not nullable.
column_value	**nvarchar(2048)**	The configured value of the column. Is nullable.
object_type	**nvarchar(60)**	The type of the object. Is not nullable. object_type is one of: event target
object_name	**nvarchar(60)**	The name of the object to which this column belongs. Is not nullable.
object_package_guid	**uniqueidentifier**	The GUID of the package that contains the object. Is not nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_xe_session_object_columns.object_name dm_xe_session_object_columns.object_package_guid	sys.dm_xe_objects.package_guid sys.dm_xe_objects.name	Many-to-one

FROM	TO	RELATIONSHIP
dm_xe_session_object_columns.column_name dm_xe_session_object_columns.column_id	sys.dm_xe_object_columns.name sys.dm_xe_object_columns.column_id	Many-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_xe_session_targets (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about session targets.

COLUMN NAME	DATA TYPE	DESCRIPTION
event_session_address	**varbinary(8)**	The memory address of the event session. Has a many-to-one relationship with sys.dm_xe_sessions.address. Is not nullable.
target_name	**nvarchar(60)**	The name of the target within a session. Is not nullable.
target_package_guid	**uniqueidentifier**	The GUID of the package that contains the target. Is not nullable.
execution_count	**bigint**	The number of times the target has been executed for the session. Is not nullable.
execution_duration_ms	**bigint**	The total amount of time, in milliseconds, that the target has been executing. Is not nullable.
target_data	**nvarchar(max)**	The data that the target maintains, such as event aggregation information. Is nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_xe_session_targets.event_session_address	sys.dm_xe_sessions.address	Many-to-one

Change History

UPDATED CONTENT
Corrected the data type for the target_data column.
Corrected the description for the target_data column to indicate that the value is nullable.

UPDATED CONTENT
Corrected the "Relationship Cardinalities" table.

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_xe_sessions (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about an active extended events session. This session is a collection of events, actions, and targets.

COLUMN NAME	DATA TYPE	DESCRIPTION
address	**varbinary(8)**	The memory address of the session. address is unique across the local system. Is not nullable.
name	**nvarchar(256)**	The name of the session. name is unique across the local system. Is not nullable.
pending_buffers	**int**	The number of full buffers that are pending processing. Is not nullable.
total_regular_buffers	**int**	The total number of regular buffers that are associated with the session. Is not nullable. Note: Regular buffers are used most of the time. These buffers are of sufficient size to hold many events. Typically, there will be three or more buffers per session. The number of regular buffers is automatically determined by the server, based on the memory partitioning that is set through the MEMORY_PARTITION_MODE option. The size of the regular buffers is equal to the value of the MAX_MEMORY option (default of 4 MB), divided by the number of buffers. For more information about the MEMORY_PARTITION_MODE and the MAX_MEMORY options, see CREATE EVENT SESSION (Transact-SQL).
regular_buffer_size	**bigint**	The regular buffer size, in bytes. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
total_large_buffers	**int**	The total number of large buffers. Is not nullable. Note: Large buffers are used when an event is larger than a regular buffer. They are set aside explicitly for this purpose. Large buffers are allocated when the event session starts, and are sized according to the MAX_EVENT_SIZE option. For more information about the MAX_EVENT_SIZE option, see CREATE EVENT SESSION (Transact-SQL).
large_buffer_size	**bigint**	The large buffer size, in bytes. Is not nullable.
total_buffer_size	**bigint**	The total size of the memory buffer that is used to store events for the session, in bytes. Is not nullable.
buffer_policy_flags	**int**	A bitmap that indicates how session event buffers behave when all the buffers are full and a new event is fired. Is not nullable.
buffer_policy_desc	**nvarchar(256)**	A description that indicates how session event buffers behave when all the buffers are full and a new event is fired. Is not nullable. buffer_policy_desc can be one of the following: Drop event Do not drop events Drop full buffer Allocate new buffer
flags	**int**	A bitmap that indicates the flags that have been set on the session. Is not nullable.
flag_desc	**nvarchar(256)**	A description of the flags set on the session. Is not nullable. flag_desc can be any combination of the following: Flush buffers on close Dedicated dispatcher Allow recursive events
dropped_event_count	**int**	The number of events that were dropped when the buffers were full. This value is **0** if the buffer policy is "Drop full buffer" or "Do not drop events". Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
dropped_buffer_count	**int**	The number of buffers that were dropped when the buffers were full. This value is **0** if the buffer policy is set to "Drop event" or "Do not drop events". Is not nullable.
blocked_event_fire_time	**int**	The length of time that event firings were blocked when buffers were full. This value is **0** if the buffer policy is "Drop full buffer" or "Drop event". Is not nullable.
create_time	**datetime**	The time that the session was created. Is not nullable.
largest_event_dropped_size	**int**	The size of the largest event that did not fit into the session buffer. Is not nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Change History

UPDATED CONTENT
Corrected the data type for the name and the blocked_event_fire_time columns.
Removed the buffer_size and total_buffers columns.
Added the total_regular_buffers,regular_buffer_size, total_large_buffers, large_buffer_size and total_buffer_size columns.

See Also

Dynamic Management Views and Functions (Transact-SQL)

Filestream and FileTable Dynamic Management Views (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section describes the dynamic management views related to the FILESTREAM and FileTable features.

Filestream Dynamic Management Views and Functions

sys.dm_filestream_file_io_handles (Transact-SQL)
Displays the currently open transactional file handles.

sys.dm_filestream_file_io_requests (Transact-SQL)
Displays current file input and file output requests.

FileTable Dynamic Management Views and Functions

sys.dm_filestream_non_transacted_handles (Transact-SQL)
Displays the currently open non-transactional file handles to FileTable data.

See Also

Filestream
Filetables
Filestream and FileTable Catalog Views (Transact-SQL)
Filestream and FileTable System Stored Procedures (Transact-SQL)

sys.dm_filestream_file_io_handles (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Displays the file handles that the Namespace Owner (NSO) knows about. Filestream handles that a client got using **OpenSqlFilestream** are displayed by this view.

COLUMN	TYPE	DESCRIPTION
handle_context_address	**varbinary(8)**	Shows the address of the internal NSO structure associated with the client's handle. Is nullable.
creation_request_id	**int**	Shows a field from the REQ_PRE_CREATE I/O request used to create this handle. Is not nullable.
creation_irp_id	**int**	Shows a field from the REQ_PRE_CREATE I/O request used to create this handle. Is not nullable
handle_id	**int**	Shows the unique ID of this handle that is assigned by the driver. Is not nullable.
creation_client_thread_id	**varbinary(8)**	Shows a field from the REQ_PRE_CREATE I/O request used to create this handle. Is nullable.
creation_client_process_id	**varbinary(8)**	Shows a field from the REQ_PRE_CREATE I/O request used to create this handle. Is nullable.
filestream_transaction_id	**varbinary(128)**	Shows the ID of the transaction associated with the given handle. This is the value returned by the **get_filestream_transaction_context** function. Use this field to join to the **sys.dm_filestream_file_io_requests** view. Is nullable.
access_type	**nvarchar(60)**	Is not nullable.
logical_path	**nvarchar(256)**	Shows the logical pathname of the file that this handle opened. This is the same pathname that is returned by the **.PathName** method of **varbinary**(**max**) filestream. Is nullable.

COLUMN	TYPE	DESCRIPTION
physical_path	**nvarchar(256)**	Shows the actual NTFS pathname of the file. This is the same pathname returned by the **.PhysicalPathName** method of the **varbinary**(**max**) filestream. It is enabled by trace flag 5556. Is nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Filestream and FileTable Dynamic Management Views (Transact-SQL)

sys.dm_filestream_file_io_requests (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Displays a list of I/O requests being processed by the Namespace Owner (NSO) at the given moment.

COLUMN	TYPE	DESCRIPTION
request_context_address	**varbinary(8)**	Shows the internal address of the NSO memory block that contains the I/O request from the driver. Is not nullable.
current_spid	**smallint**	Shows the system process id (SPID) for the current SQL Server's connection. Is not nullable.
request_type	**nvarchar(60)**	Shows the I/O request packet (IRP) type. The possible request types are REQ_PRE_CREATE, REQ_POST_CREATE, REQ_RESOLVE_VOLUME, REQ_GET_VOLUME_INFO, REQ_GET_LOGICAL_NAME, REQ_GET_PHYSICAL_NAME, REQ_PRE_CLEANUP, REQ_POST_CLEANUP, REQ_CLOSE, REQ_FSCTL, REQ_QUERY_INFO, REQ_SET_INFO, REQ_ENUM_DIRECTORY, REQ_QUERY_SECURITY, and REQ_SET_SECURITY. Is not nullable
request_state	**nvarchar(60)**	Shows the state of the I/O request in NSO. Possible values are REQ_STATE_RECEIVED, REQ_STATE_INITIALIZED, REQ_STATE_ENQUEUED, REQ_STATE_PROCESSING, REQ_STATE_FORMATTING_RESPONSE, REQ_STATE_SENDING_RESPONSE, REQ_STATE_COMPLETING, and REQ_STATE_COMPLETED. Is not nullable.
request_id	**int**	Shows the unique request ID assigned by the driver to this request. Is not nullable.
irp_id	**int**	Shows the unique IRP ID. This is useful for identifying all I/O requests related to the given IRP. Is not nullable.

COLUMN	TYPE	DESCRIPTION
handle_id	**int**	Indicated the namespace handle ID. This is the NSO specific identifier and is unique across an instance. Is not nullable.
client_thread_id	**varbinary(8)**	Shows the client application's thread ID that originates the request. ** **Warning** ** This is meaningful only if the client application is running on the same machine as SQL Server. When the client application is running remotely, the **client_thread_id** shows the thread ID of some system process that works on behalf of the remote client. Is nullable.
client_process_id	**varbinary(8)**	Shows the process ID of the client application if the client application runs on the same machine as SQL Server. For a remote client, this shows the system process ID that is working on behalf of the client application. Is nullable.
handle_context_address	**varbinary(8)**	Shows the address of the internal NSO structure associated with the client's handle. Is nullable.
filestream_transaction_id	**varbinary(128)**	Shows the ID of the transaction associated with the given handle and all the requests associated with this handle. It is the value returned by the **get_filestream_transaction_context** function. Is nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Filestream and FileTable Dynamic Management Views (Transact-SQL)

sys.dm_filestream_non_transacted_handles (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Displays the currently open non-transactional file handles associated with FileTable data.

This view contains one row per open file handle. Because the data in this view corresponds to the live internal state of the server, the data is constantly changing as handles are opened and closed. This view does not contain historical information.

For more information, see Manage FileTables.

COLUMN	TYPE	DESCRIPTION
database_id	int	ID of the database associated with the handle.
object_id	int	Object ID of the FileTable the handle is associated with.
handle_id	int	Unique handle context identifier. Used by the sp_kill_filestream_non_transacted_handles (Transact-SQL) stored procedure to kill a specific handle.
file_object_type	int	Type of the handle. This indicates the level of the hierarchy the handle was opened against, ie. database or item.
file_object_type_desc	nvarchar(120)	“UNDEFINED", “SERVER_ROOT", “DATABASE_ROOT", “TABLE_ROOT", “TABLE_ITEM"
correlation_process_id	varbinary(8)	Contains a unique identifier for the process that originated the request.
correlation_thread_id	varbinary(8)	Contains a unique identifier for the thread that originated the request.
file_context	varbinary(8)	Pointer to the file object used by this handle.
state	int	Current state of the handle. May be active, closed or killed.

COLUMN	TYPE	DESCRIPTION
state_desc	nvarchar(120)	"ACTIVE", "CLOSED", "KILLED"
current_workitem_type	int	State this handle is currently being processed by.
current_workitem_type_desc	nvarchar(120)	"NoSetWorkItemType", "FFtPreCreateWorkitem", "FFtGetPhysicalFileNameWorkitem", "FFtPostCreateWorkitem", "FFtPreCleanupWorkitem", "FFtPostCleanupWorkitem", "FFtPreCloseWorkitem", "FFtQueryDirectoryWorkItem", "FFtQueryInfoWorkItem", "FFtQueryVolumeInfoWorkItem", "FFtSetInfoWorkitem", "FFtWriteCompletionWorkitem"
fcb_id	bigint	FileTable File Control Block ID.
item_id	varbinary(892)	The Item ID for a file or directory. May be null for server root handles.
is_directory	bit	Is this a directory.
item_name	nvarchar(512)	Name of the item.
opened_file_name	nvarchar(512)	Originally requested path to be opened.
database_directory_name	nvarchar(512)	Portion of the opened_file_name that represents the database directory name.
table_directory_name	nvarchar(512)	Portion of the opened_file_name that represents the table directory name.
remaining_file_name	nvarchar(512)	Portion of the opened_file_name that represents the remaining directory name.
open_time	datetime	Time the handle was opened.
flags	int	ShareFlagsUpdatedToFcb = 0x1, DeleteOnClose = 0x2, NewFile = 0x4, PostCreateDoneForNewFile = 0x8, StreamFileOverwritten = 0x10, RequestCancelled = 0x20, NewFileCreationRolledBack = 0x40
login_id	int	ID of the principal that opened the handle.

COLUMN	TYPE	DESCRIPTION
login_name	nvarchar(512)	Name of the principal that opened the handle.
login_sid	varbinary(85)	SID of the principal that opened the handle.
read_access	bit	Opened for read access.
write_access	bit	Opened for write access.
delete_access	bit	Opened for delete access.
share_read	bit	Opened with share_read allowed.
share_write	bit	Opened with share_write allowed.
share_delete	bit	Opened with share_delete allowed.

See Also

Manage FileTables

Full-Text and Semantic Search Dynamic Management Views - Functions

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management views and functions that are related to full-text search and semantic search.

Full-Text Search Dynamic Management Views and Functions

sys.dm_fts_active_catalogs (Transact-SQL)
Returns information on the full-text catalogs that have some population activity in progress on the server.

sys.dm_fts_fdhosts
Returns information on the current activity of the filter daemon host or hosts on the server instance.

sys.dm_fts_index_keywords
Returns information about the content of a full-text index for the specified table.

sys.dm_fts_index_keywords_by_document
Returns information about the document-level content of a full-text index for the specified table. A given keyword can appear in several documents.

sys.dm_fts_index_keywords_by_property
Returns all property-related content in the full-text index of a given table. This includes all data that belongs to any property registered by the search property list associated with that full-text index.

sys.dm_fts_index_keywords_position_by_document
Returns the position of keywords in a document.

sys.dm_fts_index_population
Returns information about the full-text index populations currently in progress.

sys.dm_fts_memory_buffers
Returns information about memory buffers belonging to a specific memory pool that are used as part of a full-text crawl or a full-text crawl range.

sys.dm_fts_memory_pools
Returns information about the shared memory pools available to the Full-Text Gatherer component for a full-text crawl or a full-text crawl range.

sys.dm_fts_outstanding_batches
Returns information about each full-text indexing batch.

sys.dm_fts_parser
Returns the final tokenization result after applying a given word breaker, thesaurus, and stoplist combination to a query string input. The output is equivalent to the output if the specified given query string were issued to the Full-Text Engine.

sys.dm_fts_population_ranges
Returns information about the specific ranges related to a full-text index population currently in progress.

Semantic Search Dynamic Management Views and Functions

sys.dm_fts_semantic_similarity_population (Transact-SQL)

Returns one row of status information about the population of the document similarity index for each similarity index in each table that has an associated semantic index.

See Also

Dynamic Management Views and Functions (Transact-SQL)
System Views (Transact-SQL)

sys.dm_fts_active_catalogs (Transact-SQL)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information on the full-text catalogs that have some population activity in progress on the server.

> **NOTE**
> The following columns will be removed in a future version of Microsoft SQL Server: is_paused, previous_status, previous_status_description, row_count_in_thousands, status, status_description, and worker_count. Avoid using these columns in new development work, and plan to modify applications that currently use any of them.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	ID of the database that contains the active full-text catalog.
catalog_id	**int**	ID of the active full-text catalog.
memory_address	**varbinary(8)**	Address of memory buffers allocated for the population activity related to this full-text catalog.
name	**nvarchar(128)**	Name of the active full-text catalog.
is_paused	**bit**	Indicates whether the population of the active full-text catalog has been paused.
status	**int**	Current state of the full-text catalog. One of the following: 0 = Initializing 1 = Ready 2 = Paused 3 = Temporary error 4 = Remount needed 5 = Shutdown 6 = Quiesced for backup 7 = Backup is done through catalog 8 = Catalog is corrupt
status_description	**nvarchar(120)**	Description of current state of the active full-text catalog.

COLUMN NAME	DATA TYPE	DESCRIPTION
previous_status	**int**	Previous state of the full-text catalog. One of the following: 0 = Initializing 1 = Ready 2 = Paused 3 = Temporary error 4 = Remount needed 5 = Shutdown 6 = Quiesced for backup 7 = Backup is done through catalog 8 = Catalog is corrupt
previous_status_description	**nvarchar(120)**	Description of previous state of the active full-text catalog.
worker_count	**int**	Number of threads currently working on this full-text catalog.
active_fts_index_count	**int**	Number of full-text indexes that are being populated.
auto_population_count	**int**	Number of tables with an auto population in progress for this full-text catalog.
manual_population_count	**int**	Number of tables with manual population in progress for this full-text catalog.
full_incremental_population_count	**int**	Number of tables with a full or incremental population in progress for this full-text catalog.
row_count_in_thousands	**int**	Estimated number of rows (in thousands) in all full-text indexes in this full-text catalog.
is_importing	**bit**	Indicates whether the full-text catalog is being imported: 1 = The catalog is being imported. 2 = The catalog is not being imported.

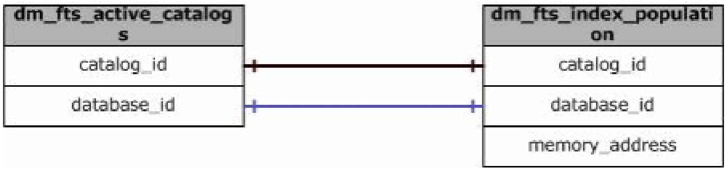
Remarks

The is_importing column was new in SQL Server 2008.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Physical Joins



Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_fts_active_catalogs.database_id	dm_fts_index_population.database_id	One-to-one
dm_fts_active_catalogs.catalog_id	dm_fts_index_population.catalog_id	One-to-one

Examples

The following example returns information about the active full-text catalogs on the current database.

```
SELECT catalog.name, catalog.is_importing, catalog.auto_population_count,
  OBJECT_NAME(population.table_id) AS table_name,
  population.population_type_description, population.is_clustered_index_scan,
  population.status_description, population.completion_type_description,
  population.queued_population_type_description, population.start_time,
  population.range_count
FROM sys.dm_fts_active_catalogs catalog
CROSS JOIN sys.dm_fts_index_population population
WHERE catalog.database_id = population.database_id
AND catalog.catalog_id = population.catalog_id
AND catalog.database_id = (SELECT dbid FROM sys.sysdatabases WHERE name = DB_NAME());
GO
```

See Also

Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)

sys.dm_fts_fdhosts (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information on the current activity of the filter daemon host or hosts on the server instance.

COLUMN NAME	DATA TYPE	DESCRIPTION
fdhost_id	**int**	ID of the filter daemon host.
fdhost_name	**nvarchar(120)**	Name of filter daemon host.
fdhost_process_id	**int**	Windows process ID of the filter daemon host.
fdhost_type	**nvarchar(120)**	Type of document being processed by the filter daemon host, one of: Single thread Multi-thread Huge document
max_thread	**int**	Maximum number of threads in the filter daemon host.
batch_count	**int**	Number of batches that are being processed in the filter daemon host.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example returns the name of the filter daemon host and the maximum number of threads in it. It also monitors how many batches are currently being processed in the filter daemon. This information can be used to diagnose performance.

```
SELECT fdhost_name, batch_count, max_thread FROM sys.dm_fts_fdhosts;
GO
```

See Also

Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)
Full-Text Search

sys.dm_fts_index_keywords (Transact-SQL)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the content of a full-text index for the specified table.

sys.dm_fts_index_keywords is a dynamic management function.

> **NOTE**
>
> To view lower-level full-text index information, use the sys.dm_fts_index_keywords_by_document dynamic management function at the document level.

Syntax

```
sys.dm_fts_index_keywords( DB_ID('database_name'), OBJECT_ID('table_name') )
```

Arguments

db_id('*database_name*')
A call to the DB_ID() function. This function accepts a database name and returns the database ID, which **sys.dm_fts_index_keywords** uses to find the specified database. If *database_name* is omitted, the current database ID is returned.

object_id('*table_name*')
A call to the OBJECT_ID() function. This function accepts a table name and returns the table ID of the table containing the full-text index to inspect.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
keyword	**nvarchar(4000)**	The hexadecimal representation of the keyword stored inside the full-text index. Note: OxFF represents the special character that indicates the end of a file or dataset.
display_term	**nvarchar(4000)**	The human-readable format of the keyword. This format is derived from the hexadecimal format. Note: The **display_term** value for OxFF is "END OF FILE."

COLUMN NAME	DATA TYPE	DESCRIPTION
column_id	**int**	ID of the column from which the current keyword was full-text indexed.
document_count	**int**	Number of documents or rows containing the current term.

Remarks

The information returned by **sys.dm_fts_index_keywords** is useful for finding out the following, among other things:

- Whether a keyword is part of the full-text index.
- How many documents or rows contain a given keyword.
- The most common keyword in the full-text index:
    - **document_count** of each *keyword_value* compared to the total **document_count**, the document count of 0xFF.
    - Typically, common keywords are likely to be appropriate to declare as stopwords.

> **NOTE**
> The **document_count** returned by **sys.dm_fts_index_keywords** may be less accurate for a specific document than the count returned by **sys.dm_fts_index_keywords_by_document** or a **CONTAINS** query. This potential inaccuracy is estimated to be less than 1%. This inaccuracy can occur because a **document_id** may be counted twice when it continues across more than one row in the index fragment, or when it appears more than once in the same row. To obtain a more accurate count for a specific document, use **sys.dm_fts_index_keywords_by_document** or a **CONTAINS** query.

Permissions

Requires membership in the **sysadmin** fixed server role.

Examples

A. Displaying high-level full-text index content

The following example displays information about the high-level content of the full-text index in the `HumanResources.JobCandidate` table.

```
SELECT * FROM sys.dm_fts_index_keywords(db_id('AdventureWorks2012'), object_id('HumanResources.JobCandidate'))
GO
```

See Also

Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)
Full-Text Search
sys.dm_fts_index_keywords_by_document (Transact-SQL)

sys.dm_fts_index_keywords_by_document (Transact-SQL)

5/3/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the document-level content of a full-text index associated with the specified table.

sys.dm_fts_index_keywords_by_document is a dynamic management function.

To view higher-level full-text index information

- sys.dm_fts_index_keywords (Transact-SQL)

  To view information about property-level content related to a document property

- sys.dm_fts_index_keywords_by_property (Transact-SQL)

Syntax

```
sys.dm_fts_index_keywords_by_document
(
    DB_ID('database_name'),     OBJECT_ID('table_name')
)
```

Arguments

db_id('*database_name*')
A call to the DB_ID() function. This function accepts a database name and returns the database ID, which sys.dm_fts_index_keywords_by_document uses to find the specified database. If *database_name* is omitted, the current database ID is returned.

object_id('*table_name*')
A call to the OBJECT_ID() function. This function accepts a table name and returns the table ID of the table containing the full-text index to inspect.

Table Returned

COLUMN	DATA TYPE	DESCRIPTION
keyword	**nvarchar(4000)**	The hexadecimal representation of the keyword that is stored inside the full-text index. Note: OxFF represents the special character that indicates the end of a file or dataset.

COLUMN	DATA TYPE	DESCRIPTION
display_term	**nvarchar(4000)**	The human-readable format of the keyword. This format is derived from the internal format that is stored in the full-text index. Note: OxFF represents the special character that indicates the end of a file or dataset.
column_id	**int**	ID of the column from which the current keyword was full-text indexed.
document_id	**int**	ID of the document or row from which the current term was full-text indexed. This ID corresponds to the full-text key value of that document or row.
occurrence_count	**int**	Number of occurrences of the current keyword in the document or row that is indicated by **document_id**. When '*search_property_name*' is specified, occurrence_count displays only the number of occurrences of the current keyword in the specified search property within the document or row.

Remarks

The information returned by sys.dm_fts_index_keywords_by_document is useful for finding out the following, among other things:

- The total number of keywords that a full-text index contains.
- Whether a keyword is part of a given document or row.
- How many times a keyword appears in the whole full-text index; that is:

  (SUM(**occurrence_count**) WHERE **keyword**=*keyword_value*)

- How many times a keyword appears in a given document or row.
- How many keywords a given document or row contains.

  Also, you can also use the information provided by sys.dm_fts_index_keywords_by_document to retrieve all the keywords belonging to a given document or row.

  When the full-text key column is an integer data type, as recommended, the document_id maps directly to the full-text key value in the base table.

  In contrast, when the full-text key column uses a non-integer data type, document_id does not represent the full-text key in the base table. In this case, to identify the row in the base table that is returned by dm_fts_index_keywords_by_document, you need to join this view with the results returned by sp_fulltext_keymappings. Before you can join them, you must store the output of the stored procedure in a temp table. Then you can join the document_id column of dm_fts_index_keywords_by_document with the DocId column that is returned by this stored procedure. Note that a **timestamp** column cannot receive values at insert time, because they are auto-generated by SQL Server. Therefore, the **timestamp** column must be converted to **varbinary(8)** columns. The following example shows these steps. In this example, *table_id* is the ID of your table, *database_name* is the name of your database, and *table_name* is the name of your table.

```
USE database_name;
GO
CREATE TABLE #MyTempTable
   (
      docid INT PRIMARY KEY ,
      [key] INT NOT NULL
   );
DECLARE @db_id int = db_id(N'database_name');
DECLARE @table_id int = OBJECT_ID(N'table_name');
INSERT INTO #MyTempTable EXEC sp_fulltext_keymappings @table_id;
SELECT * FROM sys.dm_fts_index_keywords_by_document
   ( @db_id, @table_id ) kbd
   INNER JOIN #MyTempTable tt ON tt.[docid]=kbd.document_id;
GO
```

Permissions

Requires SELECT permission on the columns covered by the full-text index and CREATE FULLTEXT CATALOG permissions.

Examples

A. Displaying full-text index content at the document level

The following example displays the content of the full-text index at the document level in the `HumanResources.JobCandidate` table of the `AdventureWorks2012` sample database.

> **NOTE**
>
> You can create this index by executing the example provided for the `HumanResources.JobCandidate` table in CREATE FULLTEXT INDEX (Transact-SQL).

```
SELECT * FROM sys.dm_fts_index_keywords_by_document(db_id('AdventureWorks'),
object_id('HumanResources.JobCandidate'));
GO
```

See Also

Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)
Full-Text Search
sys.dm_fts_index_keywords (Transact-SQL)
sys.dm_fts_index_keywords_by_property (Transact-SQL)
sp_fulltext_keymappings (Transact-SQL)
Improve the Performance of Full-Text Indexes

sys.dm_fts_index_keywords_by_property (Transact-SQL)

5/3/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns all property-related content in the full-text index of a given table. This includes all data that belongs to any property registered by the search property list associated with that full-text index.

sys.dm_fts_index_keywords_by_property is a dynamic management function that enables you to see what registered properties were emitted by IFilters at index time, as well as the exact content of every property in each indexed document.

To view all document-level content (including property-related content)

- sys.dm_fts_index_keywords_by_document (Transact-SQL)

To view higher-level full-text index information

- sys.dm_fts_index_keywords (Transact-SQL)

> **NOTE**
> For information about search property lists, see Search Document Properties with Search Property Lists.

Syntax

```
sys.dm_fts_index_keywords_by_property
(
    DB_ID('database_name'),
OBJECT_ID('table_name')
)
```

Arguments

db_id('*database_name*')
A call to the DB_ID() function. This function accepts a database name and returns the database ID, which sys.dm_fts_index_keywords_by_property uses to find the specified database. If *database_name* is omitted, the current database ID is returned.

object_id('*table_name*')
A call to the OBJECT_ID() function. This function accepts a table name and returns the table ID of the table containing the full-text index to inspect.

Table Returned

COLUMN	DATA TYPE	DESCRIPTION
keyword	**nvarchar(4000)**	The hexadecimal representation of the keyword that is stored inside the full-text index. Note: OxFF represents the special character that indicates the end of a file or dataset.
display_term	**nvarchar(4000)**	The human-readable format of the keyword. This format is derived from the internal format that is stored in the full-text index. Note: OxFF represents the special character that indicates the end of a file or dataset.
column_id	**int**	ID of the column from which the current keyword was full-text indexed.
document_id	**int**	ID of the document or row from which the current term was full-text indexed. This ID corresponds to the full-text key value of that document or row.
property_id	**int**	Internal property ID of the search property within the full-text index of the table that you specified in the OBJECT_ID('*table_name*') parameter. When a given property is added to a search property list, the Full-Text Engine registers the property and assigns it an internal property ID that is specific to that property list. The internal property ID, which is an integer, is unique to a given search property list. If a given property is registered for multiple search property lists, a different internal property ID might be assigned for each search property list. Note: The internal property ID is distinct from the property integer identifier that is specified when adding the property to the search property list. For more information, see Search Document Properties with Search Property Lists. To view the association between property_id and the property name: sys.registered_search_properties (Transact-SQL)

Remarks

This dynamic management view can answer questions such as the following:

- What content is stored on a given property for a given DocID?
- How common is a given property among the indexed documents?
- What documents actually contain a given property? This is useful if querying on a given search property does not return a document that you expected to find.

  When the full-text key column is an integer data type, as recommended, the document_id maps directly to the full-text key value in the base table.

  In contrast, when the full-text key column uses a non-integer data type, document_id does not represent the full-text key in the base table. In this case, to identify the row in the base table that is returned by dm_fts_index_keywords_by_property, you need to join this view with the results returned by sp_fulltext_keymappings. Before you can join them, you must store the output of the stored procedure in a temp table. Then you can join the document_id column of dm_fts_index_keywords_by_property with the DocId column that is returned by this stored procedure. Note that a **timestamp** column cannot receive values at insert time, because they are auto-generated by SQL Server. Therefore, the **timestamp** column must be converted to **varbinary(8)** columns. The following example shows these steps. In this example, *table_id* is the ID of your table, *database_name* is the name of your database, and *table_name* is the name of your table.

```
USE database_name;
GO
CREATE TABLE #MyTempTable
   (
      docid INT PRIMARY KEY ,
      [key] INT NOT NULL
   );
DECLARE @db_id int = db_id(N'database_name');
DECLARE @table_id int = OBJECT_ID(N'table_name');
INSERT INTO #MyTempTable EXEC sp_fulltext_keymappings @table_id;
SELECT * FROM sys.dm_fts_index_keywords_by_property
   ( @db_id, @table_id ) kbd
   INNER JOIN #MyTempTable tt ON tt.[docid]=kbd.document_id;
GO
```

Permissions

Requires SELECT permission on the columns covered by the full-text index and CREATE FULLTEXT CATALOG permissions.

Examples

The following example returns keywords from the `Author` property in the full-text index of the `Production.Document` table of the `AdventureWorks` sample database. The example uses the alias `KWBPOP` for the table returned by **sys.dm_fts_index_keywords_by_property**. The example uses inner joins to combine columns from sys.registered_search_properties and sys.fulltext_indexes.

```
-- Once the full-text index is configured to support property searching
-- on the Author property, return any keywords indexed for this property.
USE AdventureWorks2012;
GO
SELECT KWBPOP.* FROM
   sys.dm_fts_index_keywords_by_property( DB_ID(),
   object_id('Production.Document') ) AS KWBPOP
   INNER JOIN
      sys.registered_search_properties AS RSP ON(
         (KWBPOP.property_id = RSP.property_id)
         AND (RSP.property_name = 'Author') )
   INNER JOIN
      sys.fulltext_indexes AS FTI ON(
         (FTI.[object_id] = object_id('Production.Document'))
         AND (RSP.property_list_id = FTI.property_list_id) );
GO
```

See Also

Full-Text Search
Improve the Performance of Full-Text Indexes
sp_fulltext_keymappings (Transact-SQL)
sys.dm_fts_index_keywords_by_document (Transact-SQL)
sys.dm_fts_index_keywords (Transact-SQL)
sys.registered_search_properties (Transact-SQL)
sys.registered_search_property_lists (Transact-SQL)
Search Document Properties with Search Property Lists

sys.dm_fts_index_keywords_position_by_document (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns keyword positional information in the indexed documents.

Syntax

```
sys.dm_fts_index_keywords_position_by_document
(
    DB_ID('database_name'),
OBJECT_ID('table_name')
)
```

Arguments

db_id('*database_name*')
A call to the DB_ID() function. This function accepts a database name and returns the database ID, which sys.dm_fts_index_keywords_position_by_document uses to find the specified database.

object_id('*table_name*')
A call to the OBJECT_ID() function. This function accepts a table name and returns the table ID of the table containing the full-text index to inspect.

Table Returned

COLUMN	DATA TYPE	DESCRIPTION
keyword	**varbinary(128)**	Binary string representing the keyword.
display_term	**nvarchar(4000)**	The human-readable format of the keyword. This format is derived from the internal format that is stored in the full-text index.
column_id	**int**	ID of the column from which the current keyword was full-text indexed.
document_id	**bigint**	ID of the document or row from which the current term was full-text indexed. This ID corresponds to the full-text key value of that document or row.
position	**int**	The position of the keyword in the document.

Remarks

Use the DMV to identify the location of indexed words in indexed documents. This DMV can be used to troubleshoot issues when **sys.dm_fts_index_keywords_by_document** indicates the words are in the full-text index, but when you run a query using those words, the document is not returned.

Permissions

Requires SELECT permission on the columns covered by the full-text index and CREATE FULLTEXT CATALOG permissions.

Examples

The following example returns keywords from the full-text index of the `Production.Document` table of the `AdventureWorks` sample database.

```
USE AdventureWorks2012;
GO

SELECT * FROM sys.dm_fts_index_keywords_position_by_document
(
    DB_ID('AdventureWorks2012'),
    OBJECT_ID('AdventureWorks2012.Production.Document')
);
GO
```

You can add a predicate on the other columns_id as in the following example query, to further isolate the locations.

```
SELECT * FROM sys.dm_fts_index_keywords_position_by_document
(
    DB_ID('AdventureWorks2012'),
    OBJECT_ID('AdventureWorks2012.Production.Document')
)
WHERE document_id = 7 AND display_term = 'performance';
```

See Also

Full-Text Search
Improve the Performance of Full-Text Indexes
Full-Text Search and Semantic Search Functions (Transact-SQL)
Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)
Full-Text Search and Semantic Search Stored Procedures (Transact-SQL)
Search Document Properties with Search Property Lists
sys.dm_fts_index_keywords_by_document (Transact-SQL)

sys.dm_fts_index_population (Transact-SQL)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the full-text index and semantic key phrase populations currently in progress in SQL Server.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	ID of the database that contains the full-text index being populated.
catalog_id	**int**	ID of the full-text catalog that contains this full-text index.
table_id	**int**	ID of the table for which the full-text index is being populated.
memory_address	**varbinary(8)**	Memory address of the internal data structure that is used to represent an active population.
population_type	**int**	Type of population. One of the following: 1 = Full population 2 = Incremental timestamp-based population 3 = Manual update of tracked changes 4 = Background update of tracked changes.
population_type_description	**nvarchar(120)**	Description for type of population.
is_clustered_index_scan	**bit**	Indicates whether the population involves a scan on the clustered index.
range_count	**int**	Number of sub-ranges into which this population has been parallelized.
completed_range_count	**int**	Number of ranges for which processing is complete.
outstanding_batch_count	**int**	Current number of outstanding batches for this population. For more information, see sys.dm_fts_outstanding_batches (Transact-SQL).

COLUMN NAME	DATA TYPE	DESCRIPTION
status	**int**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Status of this Population. Note: some of the states are transient. One of the following: 3 = Starting 5 = Processing normally 7 = Has stopped processing For example, this status occurs when an auto merge is in progress. 11 = Population aborted 12 = Processing a semantic similarity extraction
status_description	**nvarchar(120)**	Description of status of the population.
completion_type	**int**	Status of how this population completed.
completion_type_description	**nvarchar(120)**	Description of the completion type.
worker_count	**int**	This value is always 0.
queued_population_type	**int**	Type of the population, based on tracked changes, which will follow the current population, if any.
queued_population_type_description	**nvarchar(120)**	Description of the population to follow, if any. For example, when CHANGE TRACKING = AUTO and the initial full population is in progress, this column would show "Auto population."
start_time	**datetime**	Time that the population started.
incremental_timestamp	**timestamp**	Represents the starting timestamp for a full population. For all other population types this value is the last committed checkpoint representing the progress of the populations.

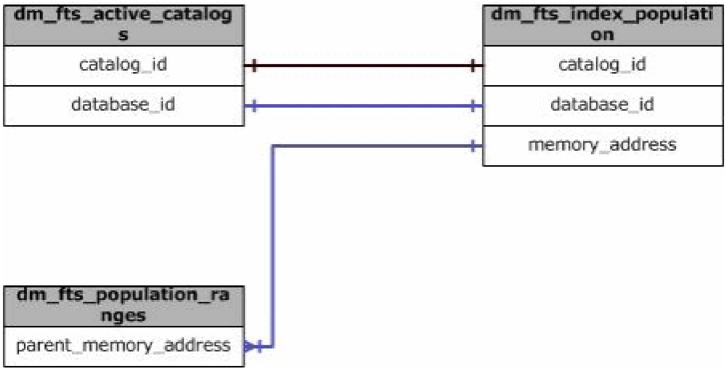
Remarks

When statistical semantic indexing is enabled in addition to full-text indexing, the semantic extraction and population of key phrases, and the extraction of document similarity data, occur simultaneously with full-text indexing. The population of the document similarity index occurs later in a second phase. For more information, see Manage and Monitor Semantic Search.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Physical Joins



Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_fts_active_catalogs.database_id	dm_fts_index_population.database_id	One-to-one
dm_fts_active_catalogs.catalog_id	dm_fts_index_population.catalog_id	One-to-one
dm_fts_population_ranges.parent_memory_address	dm_fts_index_population.memory_address	Many-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)
Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)

sys.dm_fts_memory_buffers (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about memory buffers belonging to a specific memory pool that are used as part of a full-text crawl or a full-text crawl range.

> **NOTE**
> The following column will be removed in a future release of Microsoft SQL Server: **row_count**. Avoid using this column in new development work, and plan to modify applications that currently use it.

COLUMN	DATA TYPE	DESCRIPTION
pool_id	**int**	ID of the allocated memory pool. 0 = Small buffers 1 = Large buffers
memory_address	**varbinary(8)**	Address of the allocated memory buffer.
name	**nvarchar(4000)**	Name of the shared memory buffer for which this allocation was made.
is_free	**bit**	Current state of memory buffer. 0 = Free 1 = Busy
row_count	**int**	Number of rows that this buffer is currently handling.
bytes_used	**int**	Amount, in bytes, of memory in use in this buffer.
percent_used	**int**	Percentage of allocated memory used.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Physical Joins

Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_fts_memory_buffers.pool_id	dm_fts_memory_pools.pool_id	Many-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)
Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)

sys.dm_fts_memory_pools (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the shared memory pools available to the Full-Text Gatherer component for a full-text crawl or a full-text crawl range.

COLUMN NAME	DATA TYPE	DESCRIPTION
pool_id	**int**	ID of the allocated memory pool. 0 = Small buffers 1 = Large buffers
buffer_size	**int**	Size of each allocated buffer in the memory pool.
min_buffer_limit	**int**	Minimum number of buffers allowed in the memory pool.
max_buffer_limit	**int**	Maximum number of buffers allowed in the memory pool.
buffer_count	**int**	Current number of shared memory buffers in the memory pool.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Physical Joins

Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_fts_memory_buffers.pool_id	dm_fts_memory_pools.pool_id	Many-to-one

Examples

The following example returns the total shared memory owned by the Microsoft Full-Text Gatherer component of the SQL Server process:

```
SELECT SUM(buffer_size * buffer_count) AS "total memory"
    FROM sys.dm_fts_memory_pools;
```

See Also

Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)

sys.dm_fts_outstanding_batches (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about each full-text indexing batch.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	ID of the database
catalog_id	**int**	ID of the full-text catalog
table_id	**int**	ID of the table ID that contains the full-text index
batch_id	**int**	Batch ID
memory_address	**varbinary(8)**	The batch object memory address
crawl_memory_address	**varbinary(8)**	Crawl object memory address (parent object)
memregion_memory_address	**varbinary(8)**	Memory region memory address of the outbound share memory of the filter daemon host (fdhost.exe)
hr_batch	**int**	Most recent error code for the batch
is_retry_batch	**bit**	Indicates whether this is a retry batch: 0 = No 1 = Yes
retry_hints	**int**	Type of retry needed for the batch: 0 = No retry 1 = Multi thread retry 2 = Single thread retry 3 = Single and multi thread retry 5 = Multi thread final retry 6 = Single thread final retry 7 = Single and multi thread final retry

COLUMN NAME	DATA TYPE	DESCRIPTION
retry_hints_description	**nvarchar(120)**	Description for the type of retry needed: NO RETRY MULTI THREAD RETRY SINGLE THREAD RETRY SINGLE AND MULTI THREAD RETRY MULTI THREAD FINAL RETRY SINGLE THREAD FINAL RETRY SINGLE AND MULTI THREAD FINAL RETRY
doc_failed	**bigint**	Number of documents that failed in the batch
batch_timestamp	**timestamp**	The timestamp value obtained when the batch was created

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example finds out how many batches are currently being processed for each table in the server instance.

```
SELECT database_id, table_id, COUNT(*) AS batch_count FROM sys.dm_fts_outstanding_batches GROUP BY
database_id, table_id ;
GO
```

See Also

Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)
Full-Text Search

sys.dm_fts_parser (Transact-SQL)

5/3/2018 • 6 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the final tokenization result after applying a given word breaker, thesaurus, and stoplist combination to a query string input. The tokenization result is equivalent to the output of the Full-Text Engine for the specified query string.

sys.dm_fts_parser is a dynamic management function.

Syntax

```
sys.dm_fts_parser('query_string', lcid, stoplist_id, accent_sensitivity)
```

Arguments

query_string
The query that you want to parse. *query_string* can be a string chain that CONTAINS syntax support. For example, you can include inflectional forms, a thesaurus, and logical operators.

lcid
Locale identifier (LCID) of the word breaker to be used for parsing *query_string*.

stoplist_id
ID of the stoplist, if any, to be used by the word breaker identified by *lcid*. *stoplist_id* is **int**. If you specify 'NULL', no stoplist is used. If you specify 0, the system STOPLIST is used.

A stoplist ID is unique within a database. To obtain the stoplist ID for a full-text index on a given table use the sys.fulltext_indexes catalog view.

accent_sensitivity
Boolean value that controls whether full-text search is sensitive or insensitive to diacritics. *accent_sensitivity* is **bit**, with one of the following values:

VALUE	ACCENT SENSITIVITY IS...
0	Insensitive Words such as "café" and "cafe" are treated identically.
1	Sensitive Words such as "café" and "cafe" are treated differently.

> **NOTE**
> To view the current setting of this value for a full-text catalog, run the following Transact-SQL statement: `SELECT fulltextcatalogproperty('` *catalog_name* `', 'AccentSensitivity');`.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
keyword	**varbinary(128)**	The hexadecimal representation of a given keyword returned by a word breaker. This representation is used to store the keyword in the full-text index. This value is not human-readable, but it is useful for relating a given keyword to output returned by other dynamic management views that return the content of a full-text index, such as sys.dm_fts_index_keywords and sys.dm_fts_index_keywords_by_document. **Note:** OxFF represents the special character that indicates the end of a file or dataset.
group_id	**int**	Contain an integer value that is useful for differentiating the logical group from which a given term was generated. For example, ' `Server AND DB OR FORMSOF(THESAURUS, DB)"` ' produces the following group_id values in English: 1: Server 2: DB 3: DB
phrase_id	**int**	Contains an integer value that is useful for differentiating the cases in which alternative forms of compound words, such as full-text, are issued by the word breaker. Sometimes, with presence of compound words ('multi-million'), alternative forms are issued by the word breaker. These alternative forms (phrases) need to be differentiated sometimes. For example, ' `multi-million` ' produces the following phrase_id values in English: 1 for `multi` 1 for `million` 2 for `multimillion`

COLUMN NAME	DATA TYPE	DESCRIPTION
occurrence	**int**	Indicates the order of each term in the parsing result. For example, for the phrase "`SQL Server query processor`" occurrence would contain the following occurrence values for the terms in the phrase, in English: 1 for `SQL` 2 for `Server` 3 for `query` 4 for `processor`
special_term	**nvarchar(4000)**	Contains information about the characteristics of the term that is being issued by the word breaker, one of: Exact match Noise word End of Sentence End of paragraph End of Chapter
display_term	**nvarchar(4000)**	Contains the human-readable form of the keyword. As with the functions designed to access the content of the full-text index, this displayed term might not be identical to the original term due to the denormalization limitation. However, it should be precise enough to help you identify it from the original input.

COLUMN NAME	DATA TYPE	DESCRIPTION
expansion_type	**int**	Contains information about the nature of the expansion of a given term, one of: 0 =Single word case 2=Inflectional expansion 4=Thesaurus expansion/replacement For example, consider a case in which the thesaurus defines run as an expansion of `jog`: <expansion> _{run} _{jog} </expansion> The term `FORMSOF (FREETEXT, run)` generates the following output: `run` with expansion_type=0 `runs` with expansion_type=2 `running` with expansion_type=2 `ran` with expansion_type=2 `jog` with expansion_type=4

source_term	**nvarchar(4000)**	The term or phrase from which a given term was generated or parsed. For example, a query on the '" `word breakers" AND stemmers'` produces the following source_term values in English:
		
		`word breakers` for the display_term `word`
		`word breakers` for the display_term `breakers`
		`stemmers` for the display_term `stemmers`

Remarks

sys.dm_fts_parser supports the syntax and features of full-text predicates, such as CONTAINS and FREETEXT, and functions, such as CONTAINSTABLE and FREETEXTTABLE.

Using Unicode for Parsing Special Characters

When you parse a query string, **sys.dm_fts_parser** uses the collation of the database to which you are connected, unless you specify the query string as Unicode. Therefore, for a non-Unicode string that contains special characters, such as ü or ç, the output might be unexpected, depending on the collation of the database. To process a query string independently of the database collation, prefix the string with `N`, that is, `N'`*query_string*`'`.

For more information, see "C. Displaying the Output of a String that Contains Special Characters," later in this topic.

When to Use sys.dm_fts_parser

sys.dm_fts_parser can be very powerful for debugging purposes. Some major usage scenarios include:

- To understand how a given word breaker treats a given input

  When a query returns unexpected results, a likely cause is the way that the word breaker is parsing and breaking the data. By using sys.dm_fts_parser, you discover the result that a word breaker passes to the full-text index. In addition, you can see which terms are stopwords, which are not searched in the full-text index. Whether a term is a stopword for a given language depends on whether it is in the stoplist specified by the *stoplist_id* value that is declared in the function.

  Note as well the accent sensitivity flag, which will allow the user to see how the word breaker will parse the input having in mind its accent sensitivity information.

- To understand how the stemmer works on a given input

  You can find out how the word breaker and the stemmer parse a query term and its stemming forms, by specifying a CONTAINS or CONTAINSTABLE query containing the following FORMSOF clause:

  ```
  FORMSOF( INFLECTIONAL, query_term )
  ```

  The results tell you what terms are being passed to the full-text index.

- To understand how the thesaurus expands or replaces all or part of the input

  You can also specify:

  ```
  FORMSOF( THESAURUS, query_term )
  ```

  The results of this query show how the word breaker and thesaurus interact for the query term. you can see the expansion or replacements from the thesaurus and identify the resulting query that is actually being issued against the full-text index.

  Note that if the user issues:

  ```
  FORMSOF( FREETEXT, query_term )
  ```

  The inflectional and Thesaurus capabilities will take place automatically.

  In addition to the preceding usage scenarios, sys.dm_fts_parser can help significantly to understand and troubleshoot many other issues with full-text query.

Permissions

Requires membership in the **sysadmin** fixed server role and access rights to the specified stoplist.

Examples

A. Displaying the output of a given word breaker for a keyword or phrase

The following example returns the output from using the English word breaker, whose LCID is 1033, and no stoplist on the following query string:

`The Microsoft business analysis`

Accent sensitivity is disabled.

```
SELECT * FROM sys.dm_fts_parser (' "The Microsoft business analysis" ', 1033, 0, 0);
```

B. Displaying the output of a given word breaker in the context of stoplist filtering

The following example returns the output from using the English word breaker, whose LCID is 1033, and an English stoplist, whose ID is 77, on the following query string:

`"The Microsoft business analysis" OR "MS revenue"`

Accent sensitivity is disabled.

```
SELECT * FROM sys.dm_fts_parser (' "The Microsoft business analysis"  OR " MS revenue" ', 1033, 77, 0);
```

C. Displaying the Output of a String that Contains Special Characters

The following example uses Unicode to parse the following French string:

`français`

The example specifies the LCID for the French language, `1036`, and the ID of a user-defined stoplist, `5`. Accent sensitivity is enabled.

```
SELECT * FROM sys.dm_fts_parser(N'français', 1036, 5, 1);
```

See Also

Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)
Full-Text Search
Configure and Manage Word Breakers and Stemmers for Search
Configure and Manage Thesaurus Files for Full-Text Search
Configure and Manage Stopwords and Stoplists for Full-Text Search
Query with Full-Text Search
Query with Full-Text Search
Securables

sys.dm_fts_population_ranges (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the specific ranges related to a full-text index population currently in progress.

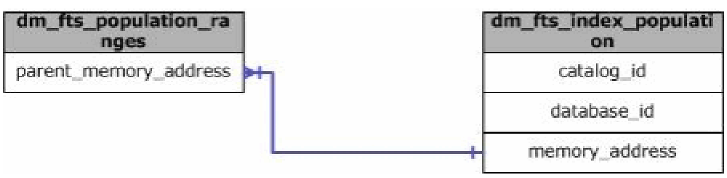
COLUMN NAME	DATA TYPE	DESCRIPTION
memory_address	**varbinary(8)**	Address of memory buffers allocated for activity related to this subrange of a full-text index population.
parent_memory_address	**varbinary(8)**	Address of memory buffers representing the parent object of all ranges of population related to a full-text index.
is_retry	**bit**	If the value is 1, this subrange is responsible for retrying rows that encountered errors.
session_id	**smallint**	ID of the session that is currently processing this task.
processed_row_count	**int**	Number of rows that have been processed by this range. Forward progress is persisted and counted every 5 minutes, rather than with every batch commit.
error_count	**int**	Number of rows that have encountered errors by this range. Forward progress is persisted and counted every 5 minutes, rather than with every batch commit.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.

On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Physical Joins



Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_fts_population_ranges.parent_memory_address	dm_fts_index_population.memory_address	Many-to-one

See Also

Full-Text Search and Semantic Search Dynamic Management Views and Functions (Transact-SQL)

sys.dm_fts_semantic_similarity_population (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row of status information about the population of the document similarity index for each similarity index in each table that has an associated semantic index.

The population step follows the extraction step. For status information about the similarity extraction step, see sys.dm_fts_index_population (Transact-SQL).

Column name	Type	Description
database_id	**int**	ID of the database that contains the full-text index being populated.
catalog_id	**int**	ID of the full-text catalog that contains this full-text index.
table_id	**int**	ID of the table for which the full-text index is being populated.
document_count	**int**	Number of total documents in the population
document_processed_count	**int**	Number of documents processed since start of this population cycle
completion_type	**int**	Status of how this population completed.
completion_type_description	**nvarchar(120)**	Description of the completion type.
worker_count	**int**	Number of worker threads associated with similarity extraction
status	**int**	Status of this Population. Note: some of the states are transient. One of the following: 3 = Starting 5 = Processing normally 7 = Has stopped processing 11 = Population aborted
status_description	**nvarchar(120)**	Description of status of the population.

start_time	**datetime**	Time that the population started.
incremental_timestamp	**timestamp**	Represents the starting timestamp for a full population. For all other population types this value is the last committed checkpoint representing the progress of the populations.

General Remarks

For more information, see Manage and Monitor Semantic Search.

Metadata

For more information about the status of semantic indexing, query sys.dm_fts_index_population (Transact-SQL).

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

Examples

The following example shows how to query the status of document similarity index populations for all tables that have an associated semantic index:

```
SELECT * FROM sys.dm_fts_semantic_similarity_population;
GO
```

See Also

Manage and Monitor Semantic Search

Geo-Replication Dynamic Management Views and Functions (Azure SQL Database)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following catalog views that display information about geo-replication.

sys.geo_replication_links (Azure SQL Database)	sys.dm_geo_replication_link_status (Azure SQL Database)
sys.dm_operation_status (Azure SQL Database)	sys.dm_continuous_copy_status (Azure SQL Database)

sys.geo_replication_links (Azure SQL Database)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Contains a row for each replication link between primary and secondary databases in a geo-replication partnership. This view resides in the logical master database.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	ID of the current database in the sys.databases view.
start_date	**datetimeoffset**	UTC time at a regional SQL Database datacenter when the database replication was initiated
modify_date	**datetimeoffset**	UTC time at regional SQL Database datacenter when the database geo-replication has completed. The new database is synchronized with the primary database as of this time. .
link_guid	**uniqueidentifier**	Unique ID of the geo-replication link.
partner_server	**sysname**	Name of the logical server containing the geo-replicated database.
partner_database	**sysname**	Name of the geo-replicated database on the linked logical server.
replication_state	**tinyint**	The state of geo-replication for this database, one of:. 0 = Pending. Creation of the active secondary database is scheduled but the necessary preparation steps are not yet completed. 1 = Seeding. The geo-replication target is being seeded but the two databases are not yet synchronized. Until seeding completes, you cannot connect to the secondary database. Removing secondary database from the primary will cancel the seeding operation. 2 = Catch-up. The secondary database is in a transactionally consistent state and is being constantly synchronized with the primary database.

COLUMN NAME	DATA TYPE	DESCRIPTION
replication_state_desc	**nvarchar(256)**	PENDING SEEDING CATCH_UP
role	**tinyint**	Geo-replication role, one of: 0 = Primary. The database_id refers to the primary database in the geo-replication partnership. 1 = Secondary. The database_id refers to the primary database in the geo-replication partnership.
role_desc	**nvarchar(256)**	PRIMARY SECONDARY
secondary_allow_connections	**tinyint**	The secondary type, one of: 0 = No. The secondary database is not accessible until failover. 1 = ReadOnly. The secondary database is accessible only to client connections with ApplicationIntent=ReadOnly. 2 = All. The secondary database is accessible to any client connection.
secondary_allow_connections _desc	**nvarchar(256)**	No All Read-Only

Permissions

This view is only available in the **master** database to the server-level principal login.

Example

Show all databases with geo-replication links.

```
SELECT
     database_id
   , start_date
   , partner_server
   , partner_database
   , replication_state
   , role_desc
   , secondary_allow_connections_desc
FROM sys.geo_replication_links;
```

See Also

ALTER DATABASE (Azure SQL Database)
sys.dm_geo_replication_link_status (Azure SQL Database)
sys.dm_operation_status (Azure SQL Database)

sys.dm_geo_replication_link_status (Azure SQL Database)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Contains a row for each replication link between primary and secondary databases in a geo-replication partnership. This includes both primary and secondary databases. If more than one continuous replication link exists for a given primary database, this table contains a row for each of the relationships. The view is created in all databases, including the logical master. However, querying this view in the logical master returns an empty set.

COLUMN NAME	DATA TYPE	DESCRIPTION
link_guid	**uniqueidentifier**	Unique ID of the replication link.
partner_server	**sysname**	Name of the logical server containing the linked database.
partner_database	**sysname**	Name of the linked database on the linked logical server.
last_replication	**datetimeoffset**	The timestamp of the last transaction's acknowledgement by the secondary based on the primary database clock. This value is available on the primary database only.
replication_lag_sec	**int**	Time difference in seconds between the last_replication value and the timestamp of that transaction's commit on the primary based on the primary database clock. This value is available on the primary database only.

COLUMN NAME	DATA TYPE	DESCRIPTION
replication_state	**tinyint**	The state of geo-replication for this database, one of:. 1 = Seeding. The geo-replication target is being seeded but the two databases are not yet synchronized. Until seeding completes, you cannot connect to the secondary database. Removing secondary database from the primary will cancel the seeding operation. 2 = Catch-up. The secondary database is in a transactionally consistent state and is being constantly synchronized with the primary database. 4 = Suspended. This is not an active continuous-copy relationship. This state usually indicates that the bandwidth available for the interlink is insufficient for the level of transaction activity on the primary database. However, the continuous-copy relationship is still intact.
replication_state_desc	**nvarchar(256)**	PENDING SEEDING CATCH_UP
role	**tinyint**	Geo-replication role, one of: 0 = Primary. The database_id refers to the primary database in the geo-replication partnership. 1 = Secondary. The database_id refers to the primary database in the geo-replication partnership.
role_desc	**nvarchar(256)**	PRIMARY SECONDARY
secondary_allow_connections	**tinyint**	The secondary type, one of: 0 = No direct connections are allowed to the secondary database and the database is not available for read access. 2 = All connections are allowed to the database in the secondary repl;ication for read-only access.
secondary_allow_connections_desc	**nvarchar(256)**	No All

COLUMN NAME	DATA TYPE	DESCRIPTION
last_commit	**datetimeoffset**	The time of last transaction committed to the database. If retrieved on the primary database, it indicates the last commit time on the primary database. If retrieved on the secondary database, it indicates the last commit time on the secondary database. If retrieved on the secondary database when the primary of the replication link is down, it indicates until what point the secondary has caught up.

NOTE

If the replication relationship is terminated by removing the secondary database (section 4.2), the row for that database in the **sys.dm_geo_replication_link_status** view disappears.

Permissions

Any account with view_database_state permission can query **sys.dm_geo_replication_link_status**.

Example

Show replication lags and last replication time of my secondary databases.

```
SELECT
     link_guid
   , partner_server
   , last_replication
   , replication_lag_sec
FROM sys.dm_geo_replication_link_status;
```

See Also

ALTER DATABASE (Azure SQL Database)
sys.geo_replication_links (Azure SQL Database)
sys.dm_operation_status (Azure SQL Database)

sys.dm_continuous_copy_status (Azure SQL Database)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each user database (V11) that is currently engaged in a Geo-replication continuous-copy relationship. If more than one continuous copy relationship is initiated for a given primary database this table contains one row for each active secondary database.

If you are using SQL Database V12 you should use sys.dm_geo_replication_link_status (because *sys.dm_continuous_copy_status* only applies to V11).

COLUMN NAME	DATA TYPE	DESCRIPTION
copy_guid	**uniqueidentifier**	Unique ID of the replica database.
partner_server	**sysname**	Name of the linked SQL Database server.
partner_database	**sysname**	Name of the linked database on the linked SQL Database server.
last_replication	**datetimeoffset**	The timestamp of the last applied replicated transaction.
replication_lag_sec	**int**	Time difference in seconds between the current time and the timestamp of the last successfully committed transaction on the primary database that has not been acknowledged by the active secondary database.

COLUMN NAME	DATA TYPE	DESCRIPTION
replication_state	**tinyint**	The state of continuous-copy replication for this database. The following are the possible values and their descriptions. 1: Seeding. The replication target is being seeded and is in a transactionally inconsistent state. Until seeding completes, you cannot connect to the active secondary database. 2: Catching up. The active secondary database is currently catching up to the primary database and is in a transactionally consistent state. 3: Re-seeding. The active secondary database is being automatically re-seeded because of an unrecoverable replication failure. 4: Suspended. This is not an active continuous-copy relationship. This state usually indicates that the bandwidth available for the interlink is insufficient for the level of transaction activity on the primary database. However, the continuous-copy relationship is still intact.
replication_state_desc	**nvarchar(256)**	Description of replication_state, one of: SEEDING CATCH_UP RE_SEEDING SUSPENDED
is_rpo_limit_reached	**bit**	This is always set to 0
is_target_role	**bit**	0 = Source of copy relationship 1 = Target of copy relationship
is_interlink_connected	**bit**	1 = Interlink is connected. 0 = Interlink is disconnected.

Permissions

To retrieve data, requires membership in the **db_owner** database role. The dbo user, members of the **dbmanager** database role, and the sa login can all query this view as well.

Remarks

The **sys.dm_continuous_copy_status** view is created in the **resource** database and is visible in all databases, including the logical master. However, querying this view in the logical master returns an empty set.

If the continuous-copy relationship is terminated on a database, the row for that database in the

sys.dm_continuous_copy_status view disappears.

Like the **sys.dm_database_copies** view, **sys.dm_continuous_copy_status** reflects the state of the continuous copy relationship in which the database is either a primary or active secondary database. Unlike **sys.dm_database_copies**, **sys.dm_continuous_copy_status** contains several columns that provide details about operations and performance. These columns include **last_replication**, and **replication_lag_sec**..

See Also

sys.dm_database_copies (Azure SQL Database)
Active Geo-Replication Stored Procedures (Transact-SQL)

Index Related Dynamic Management Views and Functions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management objects.

sys.dm_db_column_store_row_group_physical_stats (Transact-SQL)	sys.dm_db_missing_index_groups (Transact-SQL)
sys.dm_db_index_operational_stats (Transact-SQL)	sys.dm_db_index_physical_stats (Transact-SQL)
sys.dm_db_index_usage_stats (Transact-SQL)	sys.dm_db_missing_index_columns (Transact-SQL)
sys.dm_db_missing_index_details (Transact-SQL)	sys.dm_db_missing_index_group_stats (Transact-SQL)

See Also

Dynamic Management Views and Functions (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)
System Views (Transact-SQL)

sys.dm_column_store_object_pool (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns counts of different types of object memory pool usage for columnstore index objects.

COLUMN NAME	DATA TYPE	DESCRIPTION
`database_id`	`int`	ID of the database. This is unique within an instance of a SQL Server database or an Azure SQL database server.
`object_id`	`int`	ID of the object. The object is one of the object_types.
`index_id`	`int`	ID of the columnstore index.
`partition_number`	`bigint`	1-based partition number within the index or heap. Every table or view has at least one partition.
`column_id`	`int`	ID of the columnstore column. This is NULL for DELETE_BITMAP.
`row_group_id`	`int`	ID of the rowgroup.
`object_type`	`smallint`	1 = COLUMN_SEGMENT 2 = COLUMN_SEGMENT_PRIMARY_DICTIONARY 3 = COLUMN_SEGMENT_SECONDARY_DICTIONARY 4 = COLUMN_SEGMENT_BULKINSERT_DICTIONARY 5 = COLUMN_SEGMENT_DELETE_BITMAP

COLUMN NAME	DATA TYPE	DESCRIPTION
`object_type_desc`	`nvarchar(60)`	COLUMN_SEGMENT – A column segment. `object_id` is the segment ID. A segment stores all the values for one column within one rowgroup. For example, if a table has 10 columns, there are 10 column segments per rowgroup. COLUMN_SEGMENT_PRIMARY_DICTIONARY – A global dictionary that contains lookup information for all of the column segments in the table. COLUMN_SEGMENT_SECONDARY_DICTIONARY - A local dictionary associated with one column. COLUMN_SEGMENT_BULKINSERT_DICTIONARY – Another representation of the global dictionary. This provides an inverse look up of value to dictionary_id. Used for creating compressed segments as part of Tuple Mover or Bulk Load. COLUMN_SEGMENT_DELETE_BITMAP – A bitmap that tracks segment deletes. There is one delete bitmap per partition.
`access_count`	`int`	Number of read or write accesses to this object.
`memory_used_in_bytes`	`bigint`	Memory used by this object in the object pool.
`object_load_time`	`datetime`	Clock-time for when object_id was brought into the object pool.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

Index Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_db_index_physical_stats (Transact-SQL)
sys.dm_db_index_operational_stats (Transact-SQL)
sys.indexes (Transact-SQL)
sys.objects (Transact-SQL)
Monitor and Tune for Performance

sys.dm_db_column_store_row_group_operational_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns current row-level I/O, locking, and access method activity for compressed rowgroups in a columnstore index. Use **sys.dm_db_column_store_row_group_operational_stats** to track the length of time a user query must wait to read or write to a compressed rowgroup or partition of a columnstore index, and identify rowgroups that are encountering significant I/O activity or hot spots.

In-memory columnstore indexes do not appear in this DMV.

COLUMN NAME	DATA TYPE	DESCRIPTION
object_id	**int**	ID of the table with the columnstore index.
index_id	**int**	ID of the columnstore index.
partition_number	**int**	1-based partition number within the index or heap.
row_group_id	**int**	ID of the rowgroup in the columnstore index. This is unique within a partition.
scan_count	**int**	Number of scans through the rowgroup since the last SQL restart.
delete_buffer_scan_count	**int**	Number of times the delete buffer was used to determine deleted rows in this rowgroup. This includes accessing the in-memory hashtable and the underlying btree.
index_scan_count	**int**	Number of times the columnstore index partition was scanned. This is the same for all rowgroups in the partition.
rowgroup_lock_count	**bigint**	Cumulative count of lock requests for this rowgroup since the last SQL restart.
rowgroup_lock_wait_count	**bigint**	Cumulative number of times the database engine waited on this rowgroup lock since the last SQL restart.
rowgroup_lock_wait_in_ms	**bigint**	Cumulative number of milliseconds the database engine waited on this rowgroup lock since the last SQL restart.

Permissions

Requires the following permissions:

- CONTROL permission on the table specified by object_id.
- VIEW DATABASE STATE permission to return information about all objects within the database, by using the object wildcard @*object_id* = NULL

  Granting VIEW DATABASE STATE allows all objects in the database to be returned, regardless of any CONTROL permissions denied on specific objects.

  Denying VIEW DATABASE STATE disallows all objects in the database to be returned, regardless of any CONTROL permissions granted on specific objects. Also, when the database wildcard @*database_id*=NULL is specified, the database is omitted.

  For more information, see Dynamic Management Views and Functions (Transact-SQL).

See Also

Dynamic Management Views and Functions (Transact-SQL)
Index Related Dynamic Management Views and Functions (Transact-SQL)
Monitor and Tune for Performance
sys.dm_db_index_physical_stats (Transact-SQL)
sys.dm_db_index_usage_stats (Transact-SQL)
sys.dm_os_latch_stats (Transact-SQL)
sys.dm_db_partition_stats (Transact-SQL)
sys.allocation_units (Transact-SQL)
sys.indexes (Transact-SQL)

sys.dm_db_column_store_row_group_physical_stats (Transact-SQL)

5/4/2018 • 5 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Provides current rowgroup-level information about all of the columnstore indexes in the current database.

This extends the catalog view sys.column_store_row_groups (Transact-SQL).

COLUMN NAME	DATA TYPE	DESCRIPTION
object_id	**int**	ID of the underlying table.
index_id	**int**	ID of this columnstore index on *object_id* table.
partition_number	**int**	ID of the table partition that holds *row_group_id*. You can use partition_number to join this DMV to sys.partitions.
row_group_id	**int**	ID of this row group. For partitioned tables, this is unique within the partition. -1 for an in-memory tail.
delta_store_hobt_id	**bigint**	The hobt_id for a row group in the delta store. NULL if row group is not in the delta store. NULL for tail of an in-memory table.
state	**tinyint**	ID number associated *state_description*. 0 = INVISIBLE 1 = OPEN 2 = CLOSED 3 = COMPRESSED 4 = TOMBSTONE COMPRESSED is the only state that applies to in-memory tables.

COLUMN NAME	DATA TYPE	DESCRIPTION
state_desc	**nvarchar(60)**	Description of the row group state: INVISIBLE –A row group that is being built. For example: A row group in the columnstore is INVISIBLE while the data is being compressed. When the compression is finished a metadata switch changes the state of the columnstore row group from INVISIBLE to COMPRESSED, and the state of the deltastore row group from CLOSED to TOMBSTONE. OPEN – A deltastore row group that is accepting new rows. An open row group is still in rowstore format and has not been compressed to columnstore format. CLOSED – A row group in the delta store that contains the maximum number of rows, and is waiting for the tuple mover process to compress it into the columnstore. COMPRESSED – A row group that is compressed with columnstore compression and stored in the columnstore. TOMBSTONE – A row group that was formerly in the deltastore and is no longer used.
total_rows	**bigint**	Number of rows physical stored in the row group. For compressed row groups, this includes the rows that are marked deleted.
deleted_rows	**bigint**	Number of rows physically stored in a compressed row group that are marked for deletion. 0 for row groups that are in the delta store.
size_in_bytes	**bigint**	Combined size, in bytes, of all the pages in this row group. This size does not include the size required to store metadata or shared dictionaries.

COLUMN NAME	DATA TYPE	DESCRIPTION
trim_reason	**tinyint**	Reason that triggered the COMPRESSED row group to have less than the maximum number of rows. 0 – UNKNOWN_UPGRADED_FROM_PREVIOUS_VERSION 1 - NO_TRIM 2 - BULKLOAD 3 – REORG 4 – DICTIONARY_SIZE 5 – MEMORY_LIMITATION 6 – RESIDUAL_ROW_GROUP 7 - STATS_MISMATCH 8 - SPILLOVER

COLUMN NAME	DATA TYPE	DESCRIPTION
trim_reason_desc	**nvarchar(60)**	Description of *trim_reason*. 0 – UNKNOWN_UPGRADED_FROM_PREVIOUS_VERSION: Occurred when upgrading from the previous version of SQL Server. 1 - NO_TRIM: The row group was not trimmed. The row group was compressed with the maximum of 1,048,476 rows. The number of rows could be less if a subsset of rows was deleted after delta rowgroup was closed 2 – BULKLOAD: The bulk load batch size limited the number of rows. 3 – REORG: Forced compression as part of REORG command. 4 – DICTIONARY_SIZE: Dictionary size grew too big to compress all of the rows together. 5 – MEMORY_LIMITATION: Not enough available memory to compress all the rows together. 6 – RESIDUAL_ROW_GROUP: Closed as part of last row group with rows < 1 million during index build operation STATS_MISMATCH: Only for columnstore on in-memory table. If stats incorrectly indicated >= 1 million qualified rows in the tail but we found fewer, the compressed rowgroup will have < 1 million rows SPILLOVER: Only for columnstore on in-memory table. If tail has > 1 million qualified rows, the last batch remaining rows are compressed if the count is between 100k and 1 million

COLUMN NAME	DATA TYPE	DESCRIPTION
transition_to_compressed_state	tinyint	Shows how this rowgroup got moved from the deltastore to a compressed state in the columnstore. 1- NOT_APPLICABLE 2 – INDEX_BUILD 3 – TUPLE_MOVER 4 – REORG_NORMAL 5 – REORG_FORCED 6 - BULKLOAD 7 - MERGE
transition_to_compressed_state_desc	nvarchar(60)	NOT_APPLICABLE – the operation does not apply to the deltastore. Or, the rowgroup was compressed prior to upgrading to SQL Server 2016 (13.x) in which case the history is not preserved. INDEX_BUILD – An index create or index rebuild compressed the rowgroup. TUPLE_MOVER – The tuple mover running in the background compressed the rowgroup. This happens after the rowgroup changes state from OPEN to CLOSED. REORG_NORMAL – The reorganization operation, ALTER INDEX ... REORG, moved the CLOSED rowgroup from the deltastore to the columnstore. This occurred before the tuple-mover had time to move the rowgroup. REORG_FORCED – This rowgroup was open in the deltastore and was forced into the columnstore before it had a full number of rows. BULKLOAD – A bulk load operation compressed the rowgroup directly without using the deltastore. MERGE – A merge operation consolidated one or more rowgroups into this rowgroup and then performed the columnstore compression.

COLUMN NAME	DATA TYPE	DESCRIPTION
has_vertipaq_optimization	bit	Vertipaq optimization improves columnstore compression by rearranging the order of the rows in the rowgroup to achieve higher compression. This optimization occurs automatically in most cases. There are two cases Vertipaq optimization is not used: a. when a delta rowgroup moves into the columnstore and there are one or more nonclustered indexes on the columnstore index - in this case Vertipaq optimization is skipped to minimizes changes to the mapping index; b. for columnstore indexes on memory-optimized tables. 0 = No 1 = Yes
generation	bigint	Row group generation associated with this row group.
created_time	datetime2	Clock time for when this rowgroup was created. NULL – for a columnstore index on an in-memory table.
closed_time	datetime2	Clock time for when this rowgroup was closed. NULL – for a columnstore index on an in-memory table.

Results

Returns one row for each rowgroup in the current database.

Permissions

Requires these permissions:

- CONTROL permission on the table.
- VIEW DATABASE STATE permission on the database.

Examples

A. Calculate fragmentaton to decide when to reorganize or rebuild a columnstore index.

For columnstore indexes, the percent of deleted rows is a good measure for the fragmentation in a rowgroup. When the fragmentation is 20% or more we recommend removing the deleted rows. For examples, see Columnstore Indexes Defragmentation.

This example joins **sys.dm_db_column_store_row_group_physical_stats** with other system tables and then calculates the `Fragmentation` column as an estimate of the efficiency of each row group in the current database. To find information on a single table remove the comment hyphens in front of the **WHERE** clause and provide a table name.

```
SELECT i.object_id,
    object_name(i.object_id) AS TableName,
    i.name AS IndexName,
    i.index_id,
    i.type_desc,
    CSRowGroups.*,
    100*(ISNULL(deleted_rows,0))/total_rows AS 'Fragmentation'
FROM sys.indexes AS i
JOIN sys.dm_db_column_store_row_group_physical_stats AS CSRowGroups
    ON i.object_id = CSRowGroups.object_id AND i.index_id = CSRowGroups.index_id
-- WHERE object_name(i.object_id) = 'table_name'
ORDER BY object_name(i.object_id), i.name, row_group_id;
```

See Also

Object Catalog Views (Transact-SQL)
Catalog Views (Transact-SQL)
Querying the SQL Server System Catalog FAQ
sys.columns (Transact-SQL)
sys.all_columns (Transact-SQL)
sys.computed_columns (Transact-SQL)
Columnstore Indexes Guide
sys.column_store_dictionaries (Transact-SQL)
sys.column_store_segments (Transact-SQL)

sys.dm_db_index_operational_stats (Transact-SQL)

5/4/2018 • 10 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns current lowore-level I/O, locking, latching, and access method activity for each partition of a table or index in the database.

Memory-optimized indexes do not appear in this DMV.

> **NOTE**
> **sys.dm_db_index_operational_stats** does not return information about memory-optimized indexes. For information about memory-optimized index use, see sys.dm_db_xtp_index_stats (Transact-SQL).

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_db_index_operational_stats (
    { database_id | NULL | 0 | DEFAULT }
  , { object_id | NULL | 0 | DEFAULT }
  , { index_id | 0 | NULL | -1 | DEFAULT }
  , { partition_number | NULL | 0 | DEFAULT }
)
```

Arguments

database_id | NULL | 0 | DEFAULT
ID of the database. *database_id* is **smallint**. Valid inputs are the ID number of a database, NULL, 0, or DEFAULT. The default is 0. NULL, 0, and DEFAULT are equivalent values in this context.

Specify NULL to return information for all databases in the instance of SQL Server. If you specify NULL for *database_id*, you must also specify NULL for *object_id*, *index_id*, and *partition_number*.

The built-in function DB_ID can be specified.

object_id | NULL | 0 | DEFAULT
Object ID of the table or view the index is on. *object_id* is **int**.

Valid inputs are the ID number of a table and view, NULL, 0, or DEFAULT. The default is 0. NULL, 0, and DEFAULT are equivalent values in this context.

Specify NULL to return cached information for all tables and views in the specified database. If you specify NULL for *object_id*, you must also specify NULL for *index_id* and *partition_number*.

index_id | 0 | NULL | -1 | DEFAULT
ID of the index. *index_id* is **int**. Valid inputs are the ID number of an index, 0 if *object_id* is a heap, NULL, -1, or DEFAULT. The default is -1, NULL, -1, and DEFAULT are equivalent values in this context.

Specify NULL to return cached information for all indexes for a base table or view. If you specify NULL for *index_id*, you must also specify NULL for *partition_number*.

partition_number | NULL | 0 | DEFAULT
Partition number in the object. *partition_number* is **int**. Valid inputs are the *partion_number* of an index or heap, NULL, 0, or DEFAULT. The default is 0. NULL, 0, and DEFAULT are equivalent values in this context.

Specify NULL to return cached information for all partitions of the index or heap.

partition_number is 1-based. A nonpartitioned index or heap has *partition_number* set to 1.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**smallint**	Database ID.
object_id	**int**	ID of the table or view.
index_id	**int**	ID of the index or heap. 0 = Heap
hobt_id	**bigint**	**Applies to**: SQL Server (SQL Server 2016 (13.x) through current version), Azure SQL Database. ID of the data heap or B-tree rowset that tracks internal data for a columnstore index. NULL – this is not an internal columnstore rowset. For more details, see sys.internal_partitions (Transact-SQL)
partition_number	**int**	1-based partition number within the index or heap.
leaf_insert_count	**bigint**	Cumulative count of leaf-level inserts.
leaf_delete_count	**bigint**	Cumulative count of leaf-level deletes. leaf_delete_count is only incremented for deleted records that are not marked as ghost first. For deleted records that are ghosted first, **leaf_ghost_count** is incremented instead.
leaf_update_count	**bigint**	Cumulative count of leaf-level updates.

COLUMN NAME	DATA TYPE	DESCRIPTION
leaf_ghost_count	**bigint**	Cumulative count of leaf-level rows that are marked as deleted, but not yet removed. This count does not include records that are immediately deleted without being marked as ghost. These rows are removed by a cleanup thread at set intervals. This value does not include rows that are retained, because of an outstanding snapshot isolation transaction.
nonleaf_insert_count	**bigint**	Cumulative count of inserts above the leaf level. 0 = Heap or columnstore
nonleaf_delete_count	**bigint**	Cumulative count of deletes above the leaf level. 0 = Heap or columnstore
nonleaf_update_count	**bigint**	Cumulative count of updates above the leaf level. 0 = Heap or columnstore
leaf_allocation_count	**bigint**	Cumulative count of leaf-level page allocations in the index or heap. For an index, a page allocation corresponds to a page split.
nonleaf_allocation_count	**bigint**	Cumulative count of page allocations caused by page splits above the leaf level. 0 = Heap or columnstore
leaf_page_merge_count	**bigint**	Cumulative count of page merges at the leaf level. Always 0 for columnstore index.
nonleaf_page_merge_count	**bigint**	Cumulative count of page merges above the leaf level. 0 = Heap or columnstore
range_scan_count	**bigint**	Cumulative count of range and table scans started on the index or heap.
singleton_lookup_count	**bigint**	Cumulative count of single row retrievals from the index or heap.
forwarded_fetch_count	**bigint**	Count of rows that were fetched through a forwarding record. 0 = Indexes

COLUMN NAME	DATA TYPE	DESCRIPTION
lob_fetch_in_pages	**bigint**	Cumulative count of large object (LOB) pages retrieved from the LOB_DATA allocation unit. These pages contain data that is stored in columns of type **text**, **ntext**, **image**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, and **xml**. For more information, see Data Types (Transact-SQL).
lob_fetch_in_bytes	**bigint**	Cumulative count of LOB data bytes retrieved.
lob_orphan_create_count	**bigint**	Cumulative count of orphan LOB values created for bulk operations. 0 = Nonclustered index
lob_orphan_insert_count	**bigint**	Cumulative count of orphan LOB values inserted during bulk operations. 0 = Nonclustered index
row_overflow_fetch_in_pages	**bigint**	Cumulative count of row-overflow data pages retrieved from the ROW_OVERFLOW_DATA allocation unit. These pages contain data stored in columns of type **varchar(n)**, **nvarchar(n)**, **varbinary(n)**, and **sql_variant** that has been pushed off-row.
row_overflow_fetch_in_bytes	**bigint**	Cumulative count of row-overflow data bytes retrieved.
column_value_push_off_row_count	**bigint**	Cumulative count of column values for LOB data and row-overflow data that is pushed off-row to make an inserted or updated row fit within a page.
column_value_pull_in_row_count	**bigint**	Cumulative count of column values for LOB data and row-overflow data that is pulled in-row. This occurs when an update operation frees up space in a record and provides an opportunity to pull in one or more off-row values from the LOB_DATA or ROW_OVERFLOW_DATA allocation units to the IN_ROW_DATA allocation unit.
row_lock_count	**bigint**	Cumulative number of row locks requested.
row_lock_wait_count	**bigint**	Cumulative number of times the Database Engine waited on a row lock.

COLUMN NAME	DATA TYPE	DESCRIPTION
row_lock_wait_in_ms	**bigint**	Total number of milliseconds the Database Engine waited on a row lock.
page_lock_count	**bigint**	Cumulative number of page locks requested.
page_lock_wait_count	**bigint**	Cumulative number of times the Database Engine waited on a page lock.
page_lock_wait_in_ms	**bigint**	Total number of milliseconds the Database Engine waited on a page lock.
index_lock_promotion_attempt_count	**bigint**	Cumulative number of times the Database Engine tried to escalate locks.
index_lock_promotion_count	**bigint**	Cumulative number of times the Database Engine escalated locks.
page_latch_wait_count	**bigint**	Cumulative number of times the Database Engine waited, because of latch contention.
page_latch_wait_in_ms	**bigint**	Cumulative number of milliseconds the Database Engine waited, because of latch contention.
page_io_latch_wait_count	**bigint**	Cumulative number of times the Database Engine waited on an I/O page latch.
page_io_latch_wait_in_ms	**bigint**	Cumulative number of milliseconds the Database Engine waited on a page I/O latch.
tree_page_latch_wait_count	**bigint**	Subset of **page_latch_wait_count** that includes only the upper-level B-tree pages. Always 0 for a heap or columnstore index.
tree_page_latch_wait_in_ms	**bigint**	Subset of **page_latch_wait_in_ms** that includes only the upper-level B-tree pages. Always 0 for a heap or columnstore index.
tree_page_io_latch_wait_count	**bigint**	Subset of **page_io_latch_wait_count** that includes only the upper-level B-tree pages. Always 0 for a heap or columnstore index.
tree_page_io_latch_wait_in_ms	**bigint**	Subset of **page_io_latch_wait_in_ms** that includes only the upper-level B-tree pages. Always 0 for a heap or columnstore index.

COLUMN NAME	DATA TYPE	DESCRIPTION
page_compression_attempt_count	**bigint**	Number of pages that were evaluated for PAGE level compression for specific partitions of a table, index, or indexed view. Includes pages that were not compressed because significant savings could not be achieved. Always 0 for columnstore index.
page_compression_success_count	**bigint**	Number of data pages that were compressed by using PAGE compression for specific partitions of a table, index, or indexed view. Always 0 for columnstore index.

Remarks

This dynamic management object does not accept correlated parameters from CROSS APPLY and OUTER APPLY.

You can use **sys.dm_db_index_operational_stats** to track the length of time that users must wait to read or write to a table, index, or partition, and identify the tables or indexes that are encountering significant I/O activity or hot spots.

Use the following columns to identify areas of contention.

To analyze a common access pattern to the table or index partition, use these columns:

- **leaf_insert_count**
- **leaf_delete_count**
- **leaf_update_count**
- **leaf_ghost_count**
- **range_scan_count**
- **singleton_lookup_count**

  To identify latching and locking contention, use these columns:

- **page_latch_wait_count** and **page_latch_wait_in_ms**

  These columns indicate whether there is latch contention on the index or heap, and the significance of the contention.

- **row_lock_count** and **page_lock_count**

  These columns indicate how many times the Database Engine tried to acquire row and page locks.

- **row_lock_wait_in_ms** and **page_lock_wait_in_ms**

  These columns indicate whether there is lock contention on the index or heap, and the significance of the contention.

  To analyze statistics of physical I/Os on an index or heap partition

- **page_io_latch_wait_count** and **page_io_latch_wait_in_ms**

  These columns indicate whether physical I/Os were issued to bring the index or heap pages into memory and how many I/Os were issued.

Column Remarks

The values in **lob_orphan_create_count** and **lob_orphan_insert_count** should always be equal.

The value in the columns **lob_fetch_in_pages** and **lob_fetch_in_bytes** can be greater than zero for nonclustered indexes that contain one or more LOB columns as included columns. For more information, see Create Indexes with Included Columns. Similarly, the value in the columns **row_overflow_fetch_in_pages** and **row_overflow_fetch_in_bytes** can be greater than 0 for nonclustered indexes if the index contains columns that can be pushed off-row.

How the Counters in the Metadata Cache Are Reset

The data returned by **sys.dm_db_index_operational_stats** exists only as long as the metadata cache object that represents the heap or index is available. This data is neither persistent nor transactionally consistent. This means you cannot use these counters to determine whether an index has been used or not, or when the index was last used. For information about this, see sys.dm_db_index_usage_stats (Transact-SQL).

The values for each column are set to zero whenever the metadata for the heap or index is brought into the metadata cache and statistics are accumulated until the cache object is removed from the metadata cache. Therefore, an active heap or index will likely always have its metadata in the cache, and the cumulative counts may reflect activity since the instance of SQL Server was last started. The metadata for a less active heap or index will move in and out of the cache as it is used. As a result, it may or may not have values available. Dropping an index will cause the corresponding statistics to be removed from memory and no longer be reported by the function. Other DDL operations against the index may cause the value of the statistics to be reset to zero.

Using System Functions to Specify Parameter Values

You can use the Transact-SQL functions DB_ID and OBJECT_ID to specify a value for the *database_id* and *object_id* parameters. However, passing values that are not valid to these functions may cause unintended results. Always make sure that a valid ID is returned when you use DB_ID or OBJECT_ID. For more information, see the Remarks section in sys.dm_db_index_physical_stats (Transact-SQL).

Permissions

Requires the following permissions:

- CONTROL permission on the specified object within the database
- VIEW DATABASE STATE permission to return information about all objects within the specified database, by using the object wildcard @*object_id* = NULL
- VIEW SERVER STATE permission to return information about all databases, by using the database wildcard @*database_id* = NULL

  Granting VIEW DATABASE STATE allows all objects in the database to be returned, regardless of any CONTROL permissions denied on specific objects.

  Denying VIEW DATABASE STATE disallows all objects in the database to be returned, regardless of any CONTROL permissions granted on specific objects. Also, when the database wildcard @*database_id*=NULL is specified, the database is omitted.

  For more information, see Dynamic Management Views and Functions (Transact-SQL).

Examples

A. Returning information for a specified table

The following example returns information for all indexes and partitions of the `Person.Address` table in the AdventureWorks2012 database. Executing this query requires, at a minimum, CONTROL permission on `Person.Address` table.

> **IMPORTANT**
>
> When you are using the Transact-SQL functions DB_ID and OBJECT_ID to return a parameter value, always ensure that a valid ID is returned. If the database or object name cannot be found, such as when they do not exist or are spelled incorrectly, both functions will return NULL. The **sys.dm_db_index_operational_stats** function interprets NULL as a wildcard value that specifies all databases or all objects. Because this can be an unintentional operation, the examples in this section demonstrate the safe way to determine database and object IDs.

```
DECLARE @db_id int;
DECLARE @object_id int;
SET @db_id = DB_ID(N'AdventureWorks2012');
SET @object_id = OBJECT_ID(N'AdventureWorks2012.Person.Address');
IF @db_id IS NULL
  BEGIN;
    PRINT N'Invalid database';
  END;
ELSE IF @object_id IS NULL
  BEGIN;
    PRINT N'Invalid object';
  END;
ELSE
  BEGIN;
    SELECT * FROM sys.dm_db_index_operational_stats(@db_id, @object_id, NULL, NULL);
  END;
GO
```

B. Returning information for all tables and indexes

The following example returns information for all tables and indexes within the instance of SQL Server. Executing this query requires VIEW SERVER STATE permission.

```
SELECT * FROM sys.dm_db_index_operational_stats( NULL, NULL, NULL, NULL);
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Index Related Dynamic Management Views and Functions (Transact-SQL)
Monitor and Tune for Performance
sys.dm_db_index_physical_stats (Transact-SQL)
sys.dm_db_index_usage_stats (Transact-SQL)
sys.dm_os_latch_stats (Transact-SQL)
sys.dm_db_partition_stats (Transact-SQL)
sys.allocation_units (Transact-SQL)
sys.indexes (Transact-SQL)

sys.dm_db_index_physical_stats (Transact-SQL)

5/4/2018 • 23 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns size and fragmentation information for the data and indexes of the specified table or view in SQL Server. For an index, one row is returned for each level of the B-tree in each partition. For a heap, one row is returned for the IN_ROW_DATA allocation unit of each partition. For large object (LOB) data, one row is returned for the LOB_DATA allocation unit of each partition. If row-overflow data exists in the table, one row is returned for the ROW_OVERFLOW_DATA allocation unit in each partition. Does not return information about xVelocity memory optimized columnstore indexes.

> **IMPORTANT**
> If you query **sys.dm_db_index_physical_stats** on a server instance that is hosting an Always On readable secondary replica, you might encounter a REDO blocking issue. This is because this dynamic management view acquires an IS lock on the specified user table or view that can block requests by a REDO thread for an X lock on that user table or view.

sys.dm_db_index_physical_stats does not return information about memory-optimized indexes. For information about memory-optimized index use, see sys.dm_db_xtp_index_stats (Transact-SQL).

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_db_index_physical_stats (
    { database_id | NULL | 0 | DEFAULT }
  , { object_id | NULL | 0 | DEFAULT }
  , { index_id | NULL | 0 | -1 | DEFAULT }
  , { partition_number | NULL | 0 | DEFAULT }
  , { mode | NULL | DEFAULT }
)
```

Arguments

database_id | NULL | 0 | DEFAULT
Is the ID of the database. *database_id* is **smallint**. Valid inputs are the ID number of a database, NULL, 0, or DEFAULT. The default is 0. NULL, 0, and DEFAULT are equivalent values in this context.

Specify NULL to return information for all databases in the instance of SQL Server. If you specify NULL for *database_id*, you must also specify NULL for *object_id*, *index_id*, and *partition_number*.

The built-in function DB_ID can be specified. When using DB_ID without specifying a database name, the compatibility level of the current database must be 90 or greater.

object_id | NULL | 0 | DEFAULT
Is the object ID of the table or view the index is on. *object_id* is **int**.

Valid inputs are the ID number of a table and view, NULL, 0, or DEFAULT. The default is 0. NULL, 0, and DEFAULT are equivalent values in this context. As of SQL Server 2016 (13.x), valid inputs also include the service broker queue name or the queue internal table name. When default parameters are applied (i.e. all objects, all indexes, etc), fragmentation information for all queues are included in the result set.

Specify NULL to return information for all tables and views in the specified database. If you specify NULL for *object_id*, you must also specify NULL for *index_id* and *partition_number*.

index_id | 0 | NULL | -1 | DEFAULT
Is the ID of the index. *index_id* is **int**. Valid inputs are the ID number of an index, 0 if *object_id* is a heap, NULL, -1, or DEFAULT. The default is -1. NULL, -1, and DEFAULT are equivalent values in this context.

Specify NULL to return information for all indexes for a base table or view. If you specify NULL for *index_id*, you must also specify NULL for *partition_number*.

partition_number | NULL | 0 | DEFAULT
Is the partition number in the object. *partition_number* is **int**. Valid inputs are the *partion_number* of an index or heap, NULL, 0, or DEFAULT. The default is 0. NULL, 0, and DEFAULT are equivalent values in this context.

Specify NULL to return information for all partitions of the owning object.

partition_number is 1-based. A nonpartitioned index or heap has *partition_number* set to 1.

mode | NULL | DEFAULT
Is the name of the mode. *mode* specifies the scan level that is used to obtain statistics. *mode* is **sysname**. Valid inputs are DEFAULT, NULL, LIMITED, SAMPLED, or DETAILED. The default (NULL) is LIMITED.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**smallint**	Database ID of the table or view.
object_id	**int**	Object ID of the table or view that the index is on.
index_id	**int**	Index ID of an index. 0 = Heap.
partition_number	**int**	1-based partition number within the owning object; a table, view, or index. 1 = Nonpartitioned index or heap.

COLUMN NAME	DATA TYPE	DESCRIPTION
index_type_desc	**nvarchar(60)**	Description of the index type: HEAP CLUSTERED INDEX NONCLUSTERED INDEX PRIMARY XML INDEX SPATIAL INDEX XML INDEX COLUMNSTORE MAPPING INDEX (internal) COLUMNSTORE DELETEBUFFER INDEX (internal) COLUMNSTORE DELETEBITMAP INDEX (internal)
hobt_id	**bigint**	Heap or B-Tree ID of the index or partition. Besides returning the hobt_id of user-defined indexes, this also returns the hobt_id of the internal columnstore indexes.
alloc_unit_type_desc	**nvarchar(60)**	Description of the allocation unit type: IN_ROW_DATA LOB_DATA ROW_OVERFLOW_DATA The LOB_DATA allocation unit contains the data that is stored in columns of type **text**, **ntext**, **image**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, and **xml**. For more information, see Data Types (Transact-SQL). The ROW_OVERFLOW_DATA allocation unit contains the data that is stored in columns of type **varchar(n)**, **nvarchar(n)**, **varbinary(n)**, and **sql_variant** that have been pushed off-row.
index_depth	**tinyint**	Number of index levels. 1 = Heap, or LOB_DATA or ROW_OVERFLOW_DATA allocation unit.

COLUMN NAME	DATA TYPE	DESCRIPTION
index_level	**tinyint**	Current level of the index. 0 for index leaf levels, heaps, and LOB_DATA or ROW_OVERFLOW_DATA allocation units. Greater than 0 for nonleaf index levels. *index_level* will be the highest at the root level of an index. The nonleaf levels of indexes are only processed when *mode* = DETAILED.
avg_fragmentation_in_percent	**float**	Logical fragmentation for indexes, or extent fragmentation for heaps in the IN_ROW_DATA allocation unit. The value is measured as a percentage and takes into account multiple files. For definitions of logical and extent fragmentation, see Remarks. 0 for LOB_DATA and ROW_OVERFLOW_DATA allocation units. NULL for heaps when *mode* = SAMPLED.
fragment_count	**bigint**	Number of fragments in the leaf level of an IN_ROW_DATA allocation unit. For more information about fragments, see Remarks. NULL for nonleaf levels of an index, and LOB_DATA or ROW_OVERFLOW_DATA allocation units. NULL for heaps when *mode* = SAMPLED.
avg_fragment_size_in_pages	**float**	Average number of pages in one fragment in the leaf level of an IN_ROW_DATA allocation unit. NULL for nonleaf levels of an index, and LOB_DATA or ROW_OVERFLOW_DATA allocation units. NULL for heaps when *mode* = SAMPLED.

COLUMN NAME	DATA TYPE	DESCRIPTION
page_count	**bigint**	Total number of index or data pages. For an index, the total number of index pages in the current level of the b-tree in the IN_ROW_DATA allocation unit. For a heap, the total number of data pages in the IN_ROW_DATA allocation unit. For LOB_DATA or ROW_OVERFLOW_DATA allocation units, total number of pages in the allocation unit.
avg_page_space_used_in_percent	**float**	Average percentage of available data storage space used in all pages. For an index, average applies to the current level of the b-tree in the IN_ROW_DATA allocation unit. For a heap, the average of all data pages in the IN_ROW_DATA allocation unit. For LOB_DATA or ROW_OVERFLOW DATA allocation units, the average of all pages in the allocation unit. NULL when *mode* = LIMITED.

COLUMN NAME	DATA TYPE	DESCRIPTION
record_count	**bigint**	Total number of records. For an index, total number of records applies to the current level of the b-tree in the IN_ROW_DATA allocation unit. For a heap, the total number of records in the IN_ROW_DATA allocation unit. **Note:** For a heap, the number of records returned from this function might not match the number of rows that are returned by running a SELECT COUNT(*) against the heap. This is because a row may contain multiple records. For example, under some update situations, a single heap row may have a forwarding record and a forwarded record as a result of the update operation. Also, most large LOB rows are split into multiple records in LOB_DATA storage. For LOB_DATA or ROW_OVERFLOW_DATA allocation units, the total number of records in the complete allocation unit. NULL when *mode* = LIMITED.
ghost_record_count	**bigint**	Number of ghost records ready for removal by the ghost cleanup task in the allocation unit. 0 for nonleaf levels of an index in the IN_ROW_DATA allocation unit. NULL when *mode* = LIMITED.
version_ghost_record_count	**bigint**	Number of ghost records retained by an outstanding snapshot isolation transaction in an allocation unit. 0 for nonleaf levels of an index in the IN_ROW_DATA allocation unit. NULL when *mode* = LIMITED.

COLUMN NAME	DATA TYPE	DESCRIPTION
min_record_size_in_bytes	**int**	Minimum record size in bytes. For an index, minimum record size applies to the current level of the b-tree in the IN_ROW_DATA allocation unit. For a heap, the minimum record size in the IN_ROW_DATA allocation unit. For LOB_DATA or ROW_OVERFLOW_DATA allocation units, the minimum record size in the complete allocation unit. NULL when *mode* = LIMITED.
max_record_size_in_bytes	**int**	Maximum record size in bytes. For an index, the maximum record size applies to the current level of the b-tree in the IN_ROW_DATA allocation unit. For a heap, the maximum record size in the IN_ROW_DATA allocation unit. For LOB_DATA or ROW_OVERFLOW_DATA allocation units, the maximum record size in the complete allocation unit. NULL when *mode* = LIMITED.
avg_record_size_in_bytes	**float**	Average record size in bytes. For an index, the average record size applies to the current level of the b-tree in the IN_ROW_DATA allocation unit. For a heap, the average record size in the IN_ROW_DATA allocation unit. For LOB_DATA or ROW_OVERFLOW_DATA allocation units, the average record size in the complete allocation unit. NULL when *mode* = LIMITED.

COLUMN NAME	DATA TYPE	DESCRIPTION
forwarded_record_count	**bigint**	Number of records in a heap that have forward pointers to another data location. (This state occurs during an update, when there is not enough room to store the new row in the original location.) NULL for any allocation unit other than the IN_ROW_DATA allocation units for a heap. NULL for heaps when *mode* = LIMITED.
compressed_page_count	**bigint**	The number of compressed pages. For heaps, newly allocated pages are not PAGE compressed. A heap is PAGE compressed under two special conditions: when data is bulk imported or when a heap is rebuilt. Typical DML operations that cause page allocations will not be PAGE compressed. Rebuild a heap when the compressed_page_count value grows larger than the threshold you want. For tables that have a clustered index, the compressed_page_count value indicates the effectiveness of PAGE compression.
hobt_id	bigint	**Applies to**: SQL Server (SQL Server 2016 (13.x) through current version), Azure SQL Database. For columnstore indexes only, this is the ID for a rowset that tracks internal columnstore data for a partition. The rowsets are stored as data heaps or binary trees. They have the same index ID as the parent columnstore index. For more information, see sys.internal_partitions (Transact-SQL). NULL if

COLUMN NAME	DATA TYPE	DESCRIPTION
column_store_delete_buffer_state	tinyint	**Applies to**: SQL Server (SQL Server 2016 (13.x) through current version), Azure SQL Database. 0 = NOT_APPLICABLE 1 = OPEN 2 = DRAINING 3 = FLUSHING 4 = RETIRING 5 = READY
column_store_delete_buff_state_desc		**Applies to**: SQL Server (SQL Server 2016 (13.x) through current version), Azure SQL Database. NOT_APPLICABLE –the parent index is not a columnstore index. OPEN – deleters and scanners use this. DRAINING – deleters are draining out but scanners still use it. FLUSHING – buffer is closed and rows in the buffer are being written to the delete bitmap. RETIRING – rows in the closed delete buffer have been written to the delete bitmap, but the buffer has not been truncated because scanners are still using it. New scanners don't need to use the retiring buffer because the open buffer is enough. READY – This delete buffer is ready for use.

Remarks

The sys.dm_db_index_physical_stats dynamic management function replaces the DBCC SHOWCONTIG statement.

Scanning Modes

The mode in which the function is executed determines the level of scanning performed to obtain the statistical data that is used by the function. *mode* is specified as LIMITED, SAMPLED, or DETAILED. The function traverses the page chains for the allocation units that make up the specified partitions of the table or index. sys.dm_db_index_physical_stats requires only an Intent-Shared (IS) table lock, regardless of the mode that it runs in.

The LIMITED mode is the fastest mode and scans the smallest number of pages. For an index, only the parent-level pages of the B-tree (that is, the pages above the leaf level) are scanned. For a heap, the associated PFS and

IAM pages are examined and the data pages of a heap are scanned in LIMITED mode.

With LIMITED mode, compressed_page_count is NULL because the Database Engine only scans non-leaf pages of the B-tree and the IAM and PFS pages of the heap. Use SAMPLED mode to get an estimated value for compressed_page_count, and use DETAILED mode to get the actual value for compressed_page_count. The SAMPLED mode returns statistics based on a 1 percent sample of all the pages in the index or heap. Results in SAMPLED mode should be regarded as approximate. If the index or heap has fewer than 10,000 pages, DETAILED mode is used instead of SAMPLED.

The DETAILED mode scans all pages and returns all statistics.

The modes are progressively slower from LIMITED to DETAILED, because more work is performed in each mode. To quickly gauge the size or fragmentation level of a table or index, use the LIMITED mode. It is the fastest and will not return a row for each nonleaf level in the IN_ROW_DATA allocation unit of the index.

Using System Functions to Specify Parameter Values

You can use the Transact-SQL functions DB_ID and OBJECT_ID to specify a value for the *database_id* and *object_id* parameters. However, passing values that are not valid to these functions may cause unintended results. For example, if the database or object name cannot be found because they do not exist or are spelled incorrectly, both functions will return NULL. The sys.dm_db_index_physical_stats function interprets NULL as a wildcard value specifying all databases or all objects.

Additionally, the OBJECT_ID function is processed before the sys.dm_db_index_physical_stats function is called and is therefore evaluated in the context of the current database, not the database specified in *database_id*. This behavior may cause the OBJECT_ID function to return a NULL value; or, if the object name exists in both the current database context and the specified database, an error message may be returned. The following examples demonstrate these unintended results.

```
USE master;
GO
-- In this example, OBJECT_ID is evaluated in the context of the master database.
-- Because Person.Address does not exist in master, the function returns NULL.
-- When NULL is specified as an object_id, all objects in the database are returned.
-- The same results are returned when an object that is not valid is specified.
SELECT * FROM sys.dm_db_index_physical_stats
    (DB_ID(N'AdventureWorks'), OBJECT_ID(N'Person.Address'), NULL, NULL , 'DETAILED');
GO
-- This example demonstrates the results of specifying a valid object name
-- that exists in both the current database context and
-- in the database specified in the database_id parameter of the
-- sys.dm_db_index_physical_stats function.
-- An error is returned because the ID value returned by OBJECT_ID does not
-- match the ID value of the object in the specified database.
CREATE DATABASE Test;
GO
USE Test;
GO
CREATE SCHEMA Person;
GO
CREATE Table Person.Address(c1 int);
GO
USE AdventureWorks2012;
GO
SELECT * FROM sys.dm_db_index_physical_stats
    (DB_ID(N'Test'), OBJECT_ID(N'Person.Address'), NULL, NULL , 'DETAILED');
GO
-- Clean up temporary database.
DROP DATABASE Test;
GO
```

Best Practice

Always make sure that a valid ID is returned when you use DB_ID or OBJECT_ID. For example, when you use OBJECT_ID, specify a three-part name such as `OBJECT_ID(N'AdventureWorks2012.Person.Address')`, or test the value returned by the functions before you use them in the sys.dm_db_index_physical_stats function. Examples A and B that follow demonstrate a safe way to specify database and object IDs.

Detecting Fragmentation

Fragmentation occurs through the process of data modifications (INSERT, UPDATE, and DELETE statements) that are made against the table and, therefore, to the indexes defined on the table. Because these modifications are not ordinarily distributed equally among the rows of the table and indexes, the fullness of each page can vary over time. For queries that scan part or all of the indexes of a table, this kind of fragmentation can cause additional page reads. This hinders parallel scanning of data.

The fragmentation level of an index or heap is shown in the avg_fragmentation_in_percent column. For heaps, the value represents the extent fragmentation of the heap. For indexes, the value represents the logical fragmentation of the index. Unlike DBCC SHOWCONTIG, the fragmentation calculation algorithms in both cases consider storage that spans multiple files and, therefore, are accurate.

Logical Fragmentation

This is the percentage of out-of-order pages in the leaf pages of an index. An out-of-order page is a page for which the next physical page allocated to the index is not the page pointed to by the next-page pointer in the current leaf page.

Extent Fragmentation

This is the percentage of out-of-order extents in the leaf pages of a heap. An out-of-order extent is one for which the extent that contains the current page for a heap is not physically the next extent after the extent that contains the previous page.

The value for avg_fragmentation_in_percent should be as close to zero as possible for maximum performance. However, values from 0 percent through 10 percent may be acceptable. All methods of reducing fragmentation, such as rebuilding, reorganizing, or re-creating, can be used to reduce these values. For more information about how to analyze the degree of fragmentation in an index, see Reorganize and Rebuild Indexes.

Reducing Fragmentation in an Index

When an index is fragmented in a way that the fragmentation is affecting query performance, there are three choices for reducing fragmentation:

- Drop and re-create the clustered index.

  Re-creating a clustered index redistributes the data and results in full data pages. The level of fullness can be configured by using the FILLFACTOR option in CREATE INDEX. The drawbacks in this method are that the index is offline during the drop and re-create cycle, and that the operation is atomic. If the index creation is interrupted, the index is not re-created. For more information, see CREATE INDEX (Transact-SQL).

- Use ALTER INDEX REORGANIZE, the replacement for DBCC INDEXDEFRAG, to reorder the leaf level pages of the index in a logical order. Because this is an online operation, the index is available while the statement is running. The operation can also be interrupted without losing work already completed. The drawback in this method is that it does not do as good a job of reorganizing the data as an index rebuild operation, and it does not update statistics.

- Use ALTER INDEX REBUILD, the replacement for DBCC DBREINDEX, to rebuild the index online or offline. For more information, see ALTER INDEX (Transact-SQL).

Fragmentation alone is not a sufficient reason to reorganize or rebuild an index. The main effect of fragmentation is that it slows down page read-ahead throughput during index scans. This causes slower response times. If the query workload on a fragmented table or index does not involve scans, because the workload is primarily singleton lookups, removing fragmentation may have no effect. For more information, see this Microsoft Web site.

> **NOTE**
>
> Running DBCC SHRINKFILE or DBCC SHRINKDATABASE may introduce fragmentation if an index is partly or completely moved during the shrink operation. Therefore, if a shrink operation must be performed, you should do it before fragmentation is removed.

Reducing Fragmentation in a Heap

To reduce the extent fragmentation of a heap, create a clustered index on the table and then drop the index. This redistributes the data while the clustered index is created. This also makes it as optimal as possible, considering the distribution of free space available in the database. When the clustered index is then dropped to re-create the heap, the data is not moved and remains optimally in position. For information about how to perform these operations, see CREATE INDEX and DROP INDEX.

Caution

Creating and dropping a clustered index on a table, rebuilds all non-clustered indexes on that table twice.

Compacting Large Object Data

By default, the ALTER INDEX REORGANIZE statement compacts pages that contain large object (LOB) data. Because LOB pages are not deallocated when empty, compacting this data can improve disk space use if lots of LOB data have been deleted, or a LOB column is dropped.

Reorganizing a specified clustered index compacts all LOB columns that are contained in the clustered index. Reorganizing a nonclustered index compacts all LOB columns that are nonkey (included) columns in the index. When ALL is specified in the statement, all indexes that are associated with the specified table or view are reorganized. Additionally, all LOB columns that are associated with the clustered index, underlying table, or nonclustered index with included columns are compacted.

Evaluating Disk Space Use

The avg_page_space_used_in_percent column indicates page fullness. To achieve optimal disk space use, this value should be close to 100 percent for an index that will not have many random inserts. However, an index that has many random inserts and has very full pages will have an increased number of page splits. This causes more fragmentation. Therefore, in order to reduce page splits, the value should be less than 100 percent. Rebuilding an index with the FILLFACTOR option specified allows the page fullness to be changed to fit the query pattern on the index. For more information about fill factor, see Specify Fill Factor for an Index. Also, ALTER INDEX REORGANIZE will compact an index by trying to fill pages to the FILLFACTOR that was last specified. This increases the value in avg_space_used_in_percent. Note that ALTER INDEX REORGANIZE cannot reduce page fullness. Instead, an index rebuild must be performed.

Evaluating Index Fragments

A fragment is made up of physically consecutive leaf pages in the same file for an allocation unit. An index has at least one fragment. The maximum fragments an index can have are equal to the number of pages in the leaf level of the index. Larger fragments mean that less disk I/O is required to read the same number of pages. Therefore, the larger the avg_fragment_size_in_pages value, the better the range scan performance. The avg_fragment_size_in_pages and avg_fragmentation_in_percent values are inversely proportional to each other. Therefore, rebuilding or reorganizing an index should reduce the amount of fragmentation and increase the fragment size.

Limitations and Restrictions

Does not return data for clustered columnstore indexes.

Permissions

Requires the following permissions:

- CONTROL permission on the specified object within the database.
- VIEW DATABASE STATE permission to return information about all objects within the specified database, by using the object wildcard @*object_id*=NULL.
- VIEW SERVER STATE permission to return information about all databases, by using the database wildcard @*database_id* = NULL.

  Granting VIEW DATABASE STATE allows all objects in the database to be returned, regardless of any CONTROL permissions denied on specific objects.

  Denying VIEW DATABASE STATE disallows all objects in the database to be returned, regardless of any CONTROL permissions granted on specific objects. Also, when the database wildcard @*database_id*=NULL is specified, the database is omitted.

  For more information, see Dynamic Management Views and Functions (Transact-SQL).

Examples

A. Returning information about a specified table

The following example returns size and fragmentation statistics for all indexes and partitions of the `Person.Address` table. The scan mode is set to `'LIMITED'` for best performance and to limit the statistics that are returned. Executing this query requires, at a minimum, CONTROL permission on the `Person.Address` table.

```
DECLARE @db_id SMALLINT;
DECLARE @object_id INT;

SET @db_id = DB_ID(N'AdventureWorks2012');
SET @object_id = OBJECT_ID(N'AdventureWorks2012.Person.Address');

IF @db_id IS NULL
BEGIN;
    PRINT N'Invalid database';
END;
ELSE IF @object_id IS NULL
BEGIN;
    PRINT N'Invalid object';
END;
ELSE
BEGIN;
    SELECT * FROM sys.dm_db_index_physical_stats(@db_id, @object_id, NULL, NULL , 'LIMITED');
END;
GO
```

B. Returning information about a heap

The following example returns all statistics for the heap `dbo.DatabaseLog` in the AdventureWorks2012 database.

Because the table contains LOB data, a row is returned for the `LOB_DATA` allocation unit in addition to the row returned for the `IN_ROW_ALLOCATION_UNIT` that is storing the data pages of the heap. Executing this query requires, at a minimum, CONTROL permission on the `dbo.DatabaseLog` table.

```
DECLARE @db_id SMALLINT;
DECLARE @object_id INT;
SET @db_id = DB_ID(N'AdventureWorks2012');
SET @object_id = OBJECT_ID(N'AdventureWorks2012.dbo.DatabaseLog');
IF @object_id IS NULL
BEGIN;
    PRINT N'Invalid object';
END;
ELSE
BEGIN;
    SELECT * FROM sys.dm_db_index_physical_stats(@db_id, @object_id, 0, NULL , 'DETAILED');
END;
GO
```

C. Returning information for all databases

The following example returns all statistics for all tables and indexes within the instance of SQL Server by specifying the wildcard `NULL` for all parameters. Executing this query requires the VIEW SERVER STATE permission.

```
SELECT * FROM sys.dm_db_index_physical_stats (NULL, NULL, NULL, NULL, NULL);
GO
```

D. Using sys.dm_db_index_physical_stats in a script to rebuild or reorganize indexes

The following example automatically reorganizes or rebuilds all partitions in a database that have an average fragmentation over 10 percent. Executing this query requires the VIEW DATABASE STATE permission. This example specifies `DB_ID` as the first parameter without specifying a database name. An error will be generated if the current database has a compatibility level of 80 or lower. To resolve the error, replace `DB_ID()` with a valid database name. For more information about database compatibility levels, see ALTER DATABASE Compatibility Level (Transact-SQL).

```
-- Ensure a USE <databasename> statement has been executed first.
SET NOCOUNT ON;
DECLARE @objectid int;
DECLARE @indexid int;
DECLARE @partitioncount bigint;
DECLARE @schemaname nvarchar(130);
DECLARE @objectname nvarchar(130);
DECLARE @indexname nvarchar(130);
DECLARE @partitionnum bigint;
DECLARE @partitions bigint;
DECLARE @frag float;
DECLARE @command nvarchar(4000);
-- Conditionally select tables and indexes from the sys.dm_db_index_physical_stats function
-- and convert object and index IDs to names.
SELECT
    object_id AS objectid,
    index_id AS indexid,
    partition_number AS partitionnum,
    avg_fragmentation_in_percent AS frag
INTO #work_to_do
FROM sys.dm_db_index_physical_stats (DB_ID(), NULL, NULL , NULL, 'LIMITED')
WHERE avg_fragmentation_in_percent > 10.0 AND index_id > 0;

-- Declare the cursor for the list of partitions to be processed.
DECLARE partitions CURSOR FOR SELECT * FROM #work_to_do;
```

```
-- Open the cursor.
OPEN partitions;

-- Loop through the partitions.
WHILE (1=1)
    BEGIN;
        FETCH NEXT
           FROM partitions
           INTO @objectid, @indexid, @partitionnum, @frag;
        IF @@FETCH_STATUS < 0 BREAK;
        SELECT @objectname = QUOTENAME(o.name), @schemaname = QUOTENAME(s.name)
        FROM sys.objects AS o
        JOIN sys.schemas as s ON s.schema_id = o.schema_id
        WHERE o.object_id = @objectid;
        SELECT @indexname = QUOTENAME(name)
        FROM sys.indexes
        WHERE  object_id = @objectid AND index_id = @indexid;
        SELECT @partitioncount = count (*)
        FROM sys.partitions
        WHERE object_id = @objectid AND index_id = @indexid;

-- 30 is an arbitrary decision point at which to switch between reorganizing and rebuilding.
        IF @frag < 30.0
            SET @command = N'ALTER INDEX ' + @indexname + N' ON ' + @schemaname + N'.' + @objectname + N'
REORGANIZE';
        IF @frag >= 30.0
            SET @command = N'ALTER INDEX ' + @indexname + N' ON ' + @schemaname + N'.' + @objectname + N'
REBUILD';
        IF @partitioncount > 1
            SET @command = @command + N' PARTITION=' + CAST(@partitionnum AS nvarchar(10));
        EXEC (@command);
        PRINT N'Executed: ' + @command;
    END;

-- Close and deallocate the cursor.
CLOSE partitions;
DEALLOCATE partitions;

-- Drop the temporary table.
DROP TABLE #work_to_do;
GO
```

E. Using sys.dm_db_index_physical_stats to show the number of page-compressed pages

The following example shows how to display and compare the total number of pages against the pages that are row and page compressed. This information can be used to determine the benefit that compression is providing for an index or table.

```
SELECT o.name,
    ips.partition_number,
    ips.index_type_desc,
    ips.record_count, ips.avg_record_size_in_bytes,
    ips.min_record_size_in_bytes,
    ips.max_record_size_in_bytes,
    ips.page_count, ips.compressed_page_count
FROM sys.dm_db_index_physical_stats ( DB_ID(), NULL, NULL, NULL, 'DETAILED') ips
JOIN sys.objects o on o.object_id = ips.object_id
ORDER BY record_count DESC;
```

F. Using sys.dm_db_index_physical_stats in SAMPLED mode

The following example shows how SAMPLED mode returns an approximate that is different than the DETAILED mode results.

```
CREATE TABLE t3 (col1 int PRIMARY KEY, col2 varchar(500)) WITH(DATA_COMPRESSION = PAGE);
GO
BEGIN TRAN
DECLARE @idx int = 0;
WHILE @idx < 1000000
BEGIN
    INSERT INTO t3 (col1, col2)
    VALUES (@idx,
    REPLICATE ('a', 100) + CAST (@idx as varchar(10)) + REPLICATE ('a', 380))
    SET @idx = @idx + 1
END
COMMIT;
GO
SELECT page_count, compressed_page_count, forwarded_record_count, *
FROM sys.dm_db_index_physical_stats (db_id(),
    object_id ('t3'), null, null, 'SAMPLED');
SELECT page_count, compressed_page_count, forwarded_record_count, *
FROM sys.dm_db_index_physical_stats (db_id(),
    object_id ('t3'), null, null, 'DETAILED');
```

G. Querying service broker queues for index fragmentation

Applies to: SQL Server 2016 (13.x) through SQL Server.

The following examples shows how to query server broker queues for fragmentation.

```
--Using queue internal table name
select * from sys.dm_db_index_physical_stats (db_id(), object_id ('sys.queue_messages_549576996'), default,
default, default)

--Using queue name directly
select * from sys.dm_db_index_physical_stats (db_id(), object_id ('ExpenseQueue'), default, default, default)
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Index Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_db_index_operational_stats (Transact-SQL)
sys.dm_db_index_usage_stats (Transact-SQL)
sys.dm_db_partition_stats (Transact-SQL)
sys.allocation_units (Transact-SQL)
System Views (Transact-SQL)

sys.dm_db_index_usage_stats (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns counts of different types of index operations and the time each type of operation was last performed.

In Azure SQL Database, dynamic management views cannot expose information that would impact database containment or expose information about other databases the user has access to. To avoid exposing this information, every row that contains data that doesn't belong to the connected tenant is filtered out.

> **NOTE**
>
> **sys.dm_db_index_usage_stats** does not return information about memory-optimized indexes. For information about memory-optimized index use, see sys.dm_db_xtp_index_stats (Transact-SQL).

> **NOTE**
>
> To call this view from Azure SQL Data Warehouse or Parallel Data Warehouse, use **sys.dm_pdw_nodes_db_index_usage_stats**.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**smallint**	ID of the database on which the table or view is defined.
object_id	**int**	ID of the table or view on which the index is defined
index_id	**int**	ID of the index.
user_seeks	**bigint**	Number of seeks by user queries.
user_scans	**bigint**	Number of scans by user queries that did not use 'seek' predicate.
user_lookups	**bigint**	Number of bookmark lookups by user queries.
user_updates	**bigint**	Number of updates by user queries. This includes Insert, Delete, and Updates representing number of operations done not the actual rows affected. For example, if you delete 1000 rows in one statement, this count increments by 1
last_user_seek	**datetime**	Time of last user seek
last_user_scan	**datetime**	Time of last user scan.

COLUMN NAME	DATA TYPE	DESCRIPTION
last_user_lookup	**datetime**	Time of last user lookup.
last_user_update	**datetime**	Time of last user update.
system_seeks	**bigint**	Number of seeks by system queries.
system_scans	**bigint**	Number of scans by system queries.
system_lookups	**bigint**	Number of lookups by system queries.
system_updates	**bigint**	Number of updates by system queries.
last_system_seek	**datetime**	Time of last system seek.
last_system_scan	**datetime**	Time of last system scan.
last_system_lookup	**datetime**	Time of last system lookup.
last_system_update	**datetime**	Time of last system update.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Remarks

Every individual seek, scan, lookup, or update on the specified index by one query execution is counted as a use of that index and increments the corresponding counter in this view. Information is reported both for operations caused by user-submitted queries, and for operations caused by internally generated queries, such as scans for gathering statistics.

The **user_updates** counter indicates the level of maintenance on the index caused by insert, update, or delete operations on the underlying table or view. You can use this view to determine which indexes are used only lightly by your applications. You can also use the view to determine which indexes are incurring maintenance overhead. You may want to consider dropping indexes that incur maintenance overhead, but are not used for queries, or are only infrequently used for queries.

The counters are initialized to empty whenever the SQL Server (MSSQLSERVER) service is started. In addition, whenever a database is detached or is shut down (for example, because AUTO_CLOSE is set to ON), all rows associated with the database are removed.

When an index is used, a row is added to **sys.dm_db_index_usage_stats** if a row does not already exist for the index. When the row is added, its counters are initially set to zero.

During upgrade to SQL Server 2008 R2, SQL Server 2012 (11.x), or SQL Server 2014 (12.x), entries in sys.dm_db_index_usage_stats are removed. Beginning with SQL Server 2016 (13.x), entries are retained as they were prior to SQL Server 2008 R2.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

Index Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_db_index_physical_stats (Transact-SQL)
sys.dm_db_index_operational_stats (Transact-SQL)
sys.indexes (Transact-SQL)
sys.objects (Transact-SQL)
Monitor and Tune for Performance

sys.dm_db_missing_index_columns (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about database table columns that are missing an index, excluding spatial indexes. **sys.dm_db_missing_index_columns** is a dynamic management function.

Syntax

```
sys.dm_db_missing_index_columns(index_handle)
```

Arguments

index_handle
An integer that uniquely identifies a missing index. It can be obtained from the following dynamic management objects:

sys.dm_db_missing_index_details (Transact-SQL)

sys.dm_db_missing_index_groups (Transact-SQL)

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
column_id	**int**	ID of the column.
column_name	**sysname**	Name of the table column.
column_usage	**varchar(20)**	How the column is used by the query. The possible values and their descriptions are: EQUALITY: Column contributes to a predicate that expresses equality, of the form: *table.column* = *constant_value* INEQUALITY: Column contributes to a predicate that expresses inequality, for example, a predicate of the form: *table.column* > *constant_value*. Any comparison operator other than "=" expresses inequality. INCLUDE: Column is not used to evaluate a predicate, but is used for another reason, for example, to cover a query.

Remarks

Information returned by **sys.dm_db_missing_index_columns** is updated when a query is optimized by the query optimizer, and is not persisted. Missing index information is kept only until SQL Server is restarted. Database administrators should periodically make backup copies of the missing index information if they want to keep it after server recycling.

Transaction Consistency

If a transaction creates or drops a table, the rows containing missing index information about the dropped objects are removed from this dynamic management object, preserving transaction consistency.

Permissions

Users must be granted the VIEW SERVER STATE permission or any permission that implies the VIEW SERVER STATE permission to query this dynamic management function.

Examples

The following example runs a query against the `Address` table and then runs a query using the `sys.dm_db_missing_index_columns` dynamic management view to return the table columns that are missing an index.

```
USE AdventureWorks2012;
GO
SELECT City, StateProvinceID, PostalCode
FROM Person.Address
WHERE StateProvinceID = 9;
GO
SELECT mig.*, statement AS table_name,
    column_id, column_name, column_usage
FROM sys.dm_db_missing_index_details AS mid
CROSS APPLY sys.dm_db_missing_index_columns (mid.index_handle)
INNER JOIN sys.dm_db_missing_index_groups AS mig ON mig.index_handle = mid.index_handle
ORDER BY mig.index_group_handle, mig.index_handle, column_id;
GO
```

See Also

sys.dm_db_missing_index_details (Transact-SQL)
sys.dm_db_missing_index_groups (Transact-SQL)
sys.dm_db_missing_index_group_stats (Transact-SQL)

sys.dm_db_missing_index_details (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns detailed information about missing indexes, excluding spatial indexes.

In Azure SQL Database, dynamic management views cannot expose information that would impact database containment or expose information about other databases the user has access to. To avoid exposing this information, every row that contains data that doesn't belong to the connected tenant is filtered out.

COLUMN NAME	DATA TYPE	DESCRIPTION
index_handle	**int**	Identifies a particular missing index. The identifier is unique across the server. **index_handle** is the key of this table.
database_id	**smallint**	Identifies the database where the table with the missing index resides.
object_id	**int**	Identifies the table where the index is missing.
equality_columns	**nvarchar(4000)**	Comma-separated list of columns that contribute to equality predicates of the form: *table.column =constant_value*
inequality_columns	**nvarchar(4000)**	Comma-separated list of columns that contribute to inequality predicates, for example, predicates of the form: *table.column > constant_value* Any comparison operator other than "=" expresses inequality.
included_columns	**nvarchar(4000)**	Comma-separated list of columns needed as covering columns for the query. For more information about covering or included columns, see Create Indexes with Included Columns. For memory-optimized indexes (both hash and memory-optimized nonclustered), ignore **included_columns**. All columns of the table are included in every memory-optimized index.
statement	**nvarchar(4000)**	Name of the table where the index is missing.

Remarks

Information returned by **sys.dm_db_missing_index_details** is updated when a query is optimized by the query optimizer, and is not persisted. Missing index information is kept only until SQL Server is restarted. Database administrators should periodically make backup copies of the missing index information if they want to keep it after server recycling.

To determine which missing index groups a particular missing index is part of, you can query the **sys.dm_db_missing_index_groups** dynamic management view by equijoining it with **sys.dm_db_missing_index_details** based on the **index_handle** column.

Using Missing Index Information in CREATE INDEX Statements

To convert the information returned by **sys.dm_db_missing_index_details** into a CREATE INDEX statement for both memory-optimized and disk-based indexes, equality columns should be put before the inequality columns, and together they should make the key of the index. Included columns should be added to the CREATE INDEX statement using the INCLUDE clause. To determine an effective order for the equality columns, order them based on their selectivity: list the most selective columns first (leftmost in the column list).

For more information about memory-optimized indexes, see Indexes for Memory-Optimized Tables.

Transaction Consistency

If a transaction creates or drops a table, the rows containing missing index information about the dropped objects are removed from this dynamic management object, preserving transaction consistency.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

sys.dm_db_missing_index_columns (Transact-SQL)
sys.dm_db_missing_index_groups (Transact-SQL)
sys.dm_db_missing_index_group_stats (Transact-SQL)

sys.dm_db_missing_index_groups (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about what missing indexes are contained in a specific missing index group, excluding spatial indexes.

In Azure SQL Database, dynamic management views cannot expose information that would impact database containment or expose information about other databases the user has access to. To avoid exposing this information, every row that contains data that doesn't belong to the connected tenant is filtered out.

COLUMN NAME	DATA TYPE	DESCRIPTION
index_group_handle	**int**	Identifies a missing index group.
index_handle	**int**	Identifies a missing index that belongs to the group specified by **index_group_handle**. An index group contains only one index.

Remarks

Information returned by **sys.dm_db_missing_index_groups** is updated when a query is optimized by the query optimizer, and is not persisted. Missing index information is kept only until SQL Server is restarted. Database administrators should periodically make backup copies of the missing index information if they want to keep it after server recycling.

Neither column of the output result set is a key, but together they form an index key.

Permissions

To query this dynamic management view, users must be granted the VIEW SERVER STATE permission or any permission that implies the VIEW SERVER STATE permission.

See Also

sys.dm_db_missing_index_columns (Transact-SQL)
sys.dm_db_missing_index_details (Transact-SQL)
sys.dm_db_missing_index_group_stats (Transact-SQL)

sys.dm_db_missing_index_group_stats (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns summary information about groups of missing indexes, excluding spatial indexes.

In Azure SQL Database, dynamic management views cannot expose information that would impact database containment or expose information about other databases the user has access to. To avoid exposing this information, every row that contains data that doesn't belong to the connected tenant is filtered out.

COLUMN NAME	DATA TYPE	DESCRIPTION
group_handle	**int**	Identifies a group of missing indexes. This identifier is unique across the server. The other columns provide information about all queries for which the index in the group is considered missing. An index group contains only one index.
unique_compiles	**bigint**	Number of compilations and recompilations that would benefit from this missing index group. Compilations and recompilations of many different queries can contribute to this column value.
user_seeks	**bigint**	Number of seeks caused by user queries that the recommended index in the group could have been used for.
user_scans	**bigint**	Number of scans caused by user queries that the recommended index in the group could have been used for.
last_user_seek	**datetime**	Date and time of last seek caused by user queries that the recommended index in the group could have been used for.
last_user_scan	**datetime**	Date and time of last scan caused by user queries that the recommended index in the group could have been used for.
avg_total_user_cost	**float**	Average cost of the user queries that could be reduced by the index in the group.

COLUMN NAME	DATA TYPE	DESCRIPTION
avg_user_impact	**float**	Average percentage benefit that user queries could experience if this missing index group was implemented. The value means that the query cost would on average drop by this percentage if this missing index group was implemented.
system_seeks	**bigint**	Number of seeks caused by system queries, such as auto stats queries, that the recommended index in the group could have been used for. For more information, see Auto Stats Event Class.
system_scans	**bigint**	Number of scans caused by system queries that the recommended index in the group could have been used for.
last_system_seek	**datetime**	Date and time of last system seek caused by system queries that the recommended index in the group could have been used for.
last_system_scan	**datetime**	Date and time of last system scan caused by system queries that the recommended index in the group could have been used for.
avg_total_system_cost	**float**	Average cost of the system queries that could be reduced by the index in the group.
avg_system_impact	**float**	Average percentage benefit that system queries could experience if this missing index group was implemented. The value means that the query cost would on average drop by this percentage if this missing index group was implemented.

Remarks

Information returned by **sys.dm_db_missing_index_group_stats** is updated by every query execution, not by every query compilation or recompilation. Usage statistics are not persisted and are kept only until SQL Server is restarted. Database administrators should periodically make backup copies of the missing index information if they want to keep the usage statistics after server recycling.

Permissions

To query this dynamic management view, users must be granted the VIEW SERVER STATE permission or any permission that implies the VIEW SERVER STATE permission.

Examples

The following examples illustrate how to use the **sys.dm_db_missing_index_group_stats** dynamic management view.

A. Find the 10 missing indexes with the highest anticipated improvement for user queries

The following query determines which 10 missing indexes would produce the highest anticipated cumulative improvement, in descending order, for user queries.

```
SELECT TOP 10 *
FROM sys.dm_db_missing_index_group_stats
ORDER BY avg_total_user_cost * avg_user_impact * (user_seeks + user_scans)DESC;
```

B. Find the individual missing indexes and their column details for a particular missing index group

The following query determines which missing indexes comprise a particular missing index group, and displays their column details. For the sake of this example, the missing index group handle is 24.

```
SELECT migs.group_handle, mid.*
FROM sys.dm_db_missing_index_group_stats AS migs
INNER JOIN sys.dm_db_missing_index_groups AS mig
    ON (migs.group_handle = mig.index_group_handle)
INNER JOIN sys.dm_db_missing_index_details AS mid
    ON (mig.index_handle = mid.index_handle)
WHERE migs.group_handle = 24;
```

This query provides the name of the database, schema, and table where an index is missing. It also provides the names of the columns that should be used for the index key. When writing the CREATE INDEX DDL statement to implement missing indexes, list equality columns first and then inequality columns in the ON <*table_name*> clause of the CREATE INDEX statement. Included columns should be listed in the INCLUDE clause of the CREATE INDEX statement. To determine an effective order for the equality columns, order them based on their selectivity, listing the most selective columns first (leftmost in the column list).

See Also

sys.dm_db_missing_index_columns (Transact-SQL)
sys.dm_db_missing_index_details (Transact-SQL)
sys.dm_db_missing_index_groups (Transact-SQL)
CREATE INDEX (Transact-SQL)

I O Related Dynamic Management Views and Functions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management objects.

sys.dm_io_backup_tapes (Transact-SQL)	sys.dm_io_cluster_shared_drives (Transact-SQL)
sys.dm_io_pending_io_requests (Transact-SQL)	sys.dm_io_virtual_file_stats (Transact-SQL)
sys.dm_io_cluster_valid_path_names (Transact-SQL)	

See Also

Dynamic Management Views and Functions (Transact-SQL)
System Views (Transact-SQL)

sys.dm_io_backup_tapes (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the list of tape devices and the status of mount requests for backups.

COLUMN NAME	DATA TYPE	DESCRIPTION
physical_device_name	**nvarchar(520)**	Name of the actual physical device on which a backup can be taken. Is not nullable.
logical_device_name	**nvarchar(256)**	User-specified name for the drive (from **sys.backup_devices**). NULL if no user-specified name is available. Is nullable.
status	**int**	Status of the tape: 1 = Open, available for use 2 = Mount pending 3 = In use 4 = Loading **Note:** While a tape is being loaded (**status = 4**), the media label is not read yet. Columns that copy media-label values, such as **media_sequence_number**, show anticipated values, which may differ from the actual values on the tape. After the label has been read, **status** changes to **3** (in use), and the media-label columns then reflect the actual tape that is loaded. Is not nullable.
status_desc	**nvarchar(520)**	Description of the tape status: AVAILABLE MOUNT PENDING IN USE LOADING MEDIA Is not nullable.
mount_request_time	**datetime**	Time at which mount was requested. NULL if no mount is pending (**status!=2**). Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
mount_expiration_time	**datetime**	Time at which mount request will expire (time-out). NULL if no mount is pending (**status!=2**). Is nullable.
database_name	**nvarchar(256)**	Database that is to be backed up onto this device. Is nullable.
spid	**int**	Session ID. This identifies the user of the tape. Is nullable.
command	**int**	Command that performs the backup. Is nullable.
command_desc	**nvarchar(120)**	Description of the command. Is nullable.
media_family_id	**int**	Index of media family (1...*n*), *n* is the number of media families in the media set. Is nullable.
media_set_name	**nvarchar(256)**	Name of the media set (if any) as specified by the MEDIANAME option when the media set was created). Is nullable.
media_set_guid	**uniqueidentifier**	Identifier that uniquely identifies the media set. Is nullable.
media_sequence_number	**int**	Index of volume within a media family (1...*n*). Is nullable.
tape_operation	**int**	Tape operation that is being performed: 1 = Read 2 = Format 3 = Init 4 = Append Is nullable.
tape_operation_desc	**nvarchar(120)**	Tape operation that is being performed: READ FORMAT INIT APPEND Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
mount_request_type	**int**	Type of the mount request: 1 = Specific tape. The tape identified by the **media_*** fields is required. 2 = Next media family. The next media family not yet restored is requested. This is used when restoring from fewer devices than there are media families. 3 = Continuation tape. The media family is being extended, and a continuation tape is requested. Is nullable.
mount_request_type_desc	**nvarchar(120)**	Type of the mount request: SPECIFIC TAPE NEXT MEDIA FAMILY CONTINUATION VOLUME Is nullable.

Permissions

The user must have VIEW SERVER STATE permission on the server.

See Also

Dynamic Management Views and Functions (Transact-SQL)
I O Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_io_cluster_shared_drives (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This view returns the drive name of each of the shared drives if the current server instance is a clustered server. If the current server instance is not a clustered instance it returns an empty rowset.

> **NOTE**
> To call this from Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_io_cluster_shared_drives**.

COLUMN NAME	DATA TYPE	DESCRIPTION
DriveName	**nchar(2)**	The name of the drive (the drive letter) that represents an individual disk taking part in the cluster shared disk array. Column is not nullable.
pdw_node_id	**int**	**Applies to**: ssPDW The identifier for the node that this distribution is on.

Remarks

When clustering is enabled, the failover cluster instance requires data and log files to be on shared disks so that they may be accessed after the instance fails over to another node. Each of the rows in this view represents a single shared disk which is used by this clustered SQL Server instance. Only disks listed by this view can be used to store data or log files for this instance of SQL Server. The disks listed in this view are those that are in the cluster resource group associated with the instance.

> **NOTE**
> This view will be deprecated in a future release. We recommend that you use sys.dm_io_cluster_valid_path_names (Transact-SQL) instead.

Permissions

The user must have VIEW SERVER STATE permission for the SQL Server instance.

Examples

The following example uses sys.dm_io_cluster_shared_drives to determine the shared drives on a clustered server instance:

```
SELECT * FROM sys.dm_io_cluster_shared_drives;
```

This is the result set:

DriveName

----

m

n

See Also

sys.dm_io_cluster_valid_path_names (Transact-SQL)
sys.dm_os_cluster_nodes (Transact-SQL)
sys.fn_servershareddrives (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)

sys.dm_io_pending_io_requests (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each pending I/O request in SQL Server.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_io_pending_io_requests**.

COLUMN NAME	DATA TYPE	DESCRIPTION
io_completion_request_address	**varbinary(8)**	Memory address of the IO request. Is not nullable.
io_type	**varchar(7)**	Type of pending I/O request. Is not nullable.
io_pending	**int**	Indicates whether the I/O request is pending or has been completed by Windows. An I/O request can still be pending even when Windows has completed the request, but SQL Server has not yet performed a context switch in which it would process the I/O request and remove it from this list. Is not nullable.
io_completion_routine_address	**varbinary(8)**	Internal function to call when the I/O request is completed. Is nullable.
io_user_data_address	**varbinary(8)**	Internal use only. Is nullable.
scheduler_address	**varbinary(8)**	Scheduler on which this I/O request was issued. The I/O request will appear on the pending I/O list of the scheduler. For more information, see sys.dm_os_schedulers (Transact-SQL). Is not nullable.
io_handle	**varbinary(8)**	File handle of the file that is used in the I/O request. Is nullable.
io_offset	**bigint**	Offset of the I/O request. Is not nullable.
io_pending_ms_ticks	**int**	Internal use only. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

Dynamic Management Views and Functions (Transact-SQL)
I O Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_io_virtual_file_stats (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns I/O statistics for data and log files. This dynamic management view replaces the fn_virtualfilestats function.

> **NOTE**
> To call this from Azure SQL Data Warehouse, use the name **sys.dm_pdw_nodes_io_virtual_file_stats**.

Syntax

```
-- Syntax for SQL Server and Azure SQL Database

sys.dm_io_virtual_file_stats (
    { database_id | NULL },
    { file_id | NULL }
)
```

```
-- Syntax for Azure SQL Data Warehouse

sys.dm_pdw_nodes_io_virtual_file_stats
```

Arguments

database_id | NULL

APPLIES TO: SQL Server (starting with 2008), Azure SQL Database

ID of the database. *database_id* is int, with no default. Valid inputs are the ID number of a database or NULL. When NULL is specified, all databases in the instance of SQL Server are returned.

The built-in function DB_ID can be specified.

file_id | NULL

APPLIES TO: SQL Server (starting with 2008), Azure SQL Database

ID of the file. *file_id* is int, with no default. Valid inputs are the ID number of a file or NULL. When NULL is specified, all files on the database are returned.

The built-in function FILE_IDEX can be specified, and refers to a file in the current database.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION

COLUMN NAME	DATA TYPE	DESCRIPTION
database_name	**sysname**	Database name. For SQL Data Warehouse, this is the name of the database stored on the node which is identified by pdw_node_id. Each node has one tempdb database that has 13 files. Each node also has one database per distribution, and each distribution database has 5 files. For example, if each node contains 4 distributions, the results show 20 distribution database files per pdw_node_id.
database_id	**smallint**	ID of database.
file_id	**smallint**	ID of file.
sample_ms	**bigint**	Number of milliseconds since the computer was started. This column can be used to compare different outputs from this function. The data type is **int** for SQL Server 2008 through SQL Server 2014 (12.x)
num_of_reads	**bigint**	Number of reads issued on the file.
num_of_bytes_read	**bigint**	Total number of bytes read on this file.
io_stall_read_ms	**bigint**	Total time, in milliseconds, that the users waited for reads issued on the file.
num_of_writes	**bigint**	Number of writes made on this file.
num_of_bytes_written	**bigint**	Total number of bytes written to the file.
io_stall_write_ms	**bigint**	Total time, in milliseconds, that users waited for writes to be completed on the file.
io_stall	**bigint**	Total time, in milliseconds, that users waited for I/O to be completed on the file.
size_on_disk_bytes	**bigint**	Number of bytes used on the disk for this file. For sparse files, this number is the actual number of bytes on the disk that are used for database snapshots.
file_handle	**varbinary**	Windows file handle for this file.

COLUMN NAME	DATA TYPE	DESCRIPTION
io_stall_queued_read_ms	**bigint**	**Does not apply to:**: SQL Server 2008 through SQL Server 2012 (11.x). Total IO latency introduced by IO resource governance for reads. Is not nullable. For more information, see sys.dm_resource_governor_resource_pools (Transact-SQL).
io_stall_queued_write_ms	**bigint**	**Does not apply to:**: SQL Server 2008 through SQL Server 2012 (11.x). Total IO latency introduced by IO resource governance for writes. Is not nullable.
pdw_node_id	**int**	**Applies to:** SQL Data Warehouse Identifier of the node for the distribution.

Permissions

Requires VIEW SERVER STATE permission. For more information, see Dynamic Management Views and Functions (Transact-SQL).

Examples

A. Return statistics for a log file

Applies to: SQL Server (starting with 2008), Azure SQL Database

The following example returns statistics for the log file in the AdventureWorks2012 database.

```
SELECT * FROM sys.dm_io_virtual_file_stats(DB_ID(N'AdventureWorks2012'), 2);
GO
```

B. Return statistics for file in tempdb

Applies to: Azure SQL Data Warehouse

```
SELECT * FROM sys.dm_pdw_nodes_io_virtual_file_stats
WHERE database_name = ‘tempdb’ AND file_id = 2;
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
I O Related Dynamic Management Views and Functions (Transact-SQL)
sys.database_files (Transact-SQL)
sys.master_files (Transact-SQL)

sys.dm_io_cluster_valid_path_names (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information on all valid shared disks, including clustered shared volumes, for a SQL Server failover cluster instance. If the instance is not clustered, an empty rowset is returned.

COLUMN NAME	DATA TYPE	DESCRIPTION
path_name	**Nvarchar(512)**	Volume mount point or drive path that can be used as a root directory for database and log files. Is not nullable.
cluster_owner_node	**Nvarchar(64)**	Current owner of the drive. For cluster shared volumes (CSV), the owner is the node which is hosting the MetaData Server. Is not nullable.
is_cluster_shared_volume	**Bit**	Returns 1 if the drive on which this path is located is a cluster shared volume; otherwise, returns 0.

Remarks

A SQL Server failover cluster instance (FCI) must use shared storage between all nodes of the FCI for data and log file storage. The disks listed in this view are those that are in the cluster resource group associated with the instance and are the only disks that can be used for data or log file storage.

> **NOTE**
> This view will replace sys.dm_io_cluster_shared_drives (Transact-SQL) in a future release.

Permissions

The user must have VIEW SERVER STATE permission for the SQL Server instance.

Examples

The following example uses sys.dm_io_cluster_valid_path_names to determine the shared drives on a clustered server instance:

```
SELECT * FROM sys.dm_io_cluster_valid_path_names;
```

See Also

sys.dm_os_cluster_nodes (Transact-SQL)
sys.dm_io_cluster_shared_drives (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following SQL Server dynamic management views (DMVs) are used with In-Memory OLTP:

For more information, see In-Memory OLTP (In-Memory Optimization).

sys.dm_db_xtp_checkpoint_stats (Transact-SQL)	sys.dm_db_xtp_checkpoint_files (Transact-SQL)
sys.dm_db_xtp_gc_cycle_stats (Transact-SQL)	sys.dm_db_xtp_hash_index_stats (Transact-SQL)
sys.dm_db_xtp_index_stats (Transact-SQL)	sys.dm_db_xtp_memory_consumers (Transact-SQL)
sys.dm_db_xtp_merge_requests (Transact-SQL)	sys.dm_db_xtp_object_stats (Transact-SQL)
sys.dm_db_xtp_nonclustered_index_stats (Transact-SQL)	sys.dm_db_xtp_table_memory_stats (Transact-SQL)
sys.dm_db_xtp_transactions (Transact-SQL)	sys.dm_xtp_gc_queue_stats (Transact-SQL)
sys.dm_xtp_gc_stats (Transact-SQL)	sys.dm_xtp_system_memory_consumers (Transact-SQL)
sys.dm_xtp_transaction_stats (Transact-SQL)	

Object Catalog Views

The following object catalog views are used specifically with In-Memory OLTP.

sys.hash_indexes (Transact-SQL)	sys.memory_optimized_tables_internal_attributes (Transact-SQL)

Internal DMVs

There are additional DMVs that are intended for internal use only, and for which we provide no direct documentation. In the area of memory-optimized tables, undocumented DMVs include the following:

- sys.dm_xtp_threads
- sys.dm_xtp_transaction_recent_rows

sys.dm_db_xtp_checkpoint_stats (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns statistics about the In-Memory OLTP checkpoint operations in the current database. If the database has no In-Memory OLTP objects, returns an empty result set.

For more information, see In-Memory OLTP (In-Memory Optimization).

```
SELECT * FROM db.sys.dm_db_xtp_checkpoint_stats;
```

SQL Server 2014 (12.x) is substantially different from more recent versions and is discussed lower in the topic at SQL Server 2014.

SQL Server 2016 (13.x) and later

The following table describes the columns in `sys.dm_db_xtp_checkpoint_stats`, beginning with **SQL Server 2016 (13.x)**.

COLUMN NAME	TYPE	DESCRIPTION
last_lsn_processed	**bigint**	Last LSN seen by the controller.
end_of_log_lsn	**numeric(38)**	The LSN of the end of log.
bytes_to_end_of_log	**bigint**	Log bytes unprocessed by the controller, corresponding to the bytes between `last_lsn_processed` and `end_of_log_lsn`.
log_consumption_rate	**bigint**	Rate of transaction log consumption by the controller (in KB/sec).
active_scan_time_in_ms	**bigint**	Time spent by the controller in actively scanning the transaction log.
total_wait_time_in_ms	**bigint**	Cumulative wait time for the controller while not scanning the log.
waits_for_io	**bigint**	Number of waits for log IO incurred by the controller thread.
io_wait_time_in_ms	**bigint**	Cumulative time spent waiting on log IO by the controller thread.
waits_for_new_log_count	**bigint**	Number of waits incurred by the controller thread for a new log to be generated.

COLUMN NAME	TYPE	DESCRIPTION
new_log_wait_time_in_ms	**bigint**	Cumulative time spent waiting on a new log by the controller thread.
idle_attempts_count	**bigint**	Number of times the controller transitioned to an idle state.
tx_segments_dispatched	**bigint**	Number of segments seen by the controller and dispatched to the serializers. Segment is a contiguous portion of log that forms a unit of serialization. It is currently sized to 1MB, but can change in future.
segment_bytes_dispatched	**bigint**	Total byte count of bytes dispatched by the controller to serializers, since the database restart.
bytes_serialized	**bigint**	Total count of bytes serialized since database restart.
serializer_user_time_in_ms	**bigint**	Time spent by serializers in user mode.
serializer_kernel_time_in_ms	**bigint**	Time spent by serializers in kernel mode.
xtp_log_bytes_consumed	**bigint**	Total count of log bytes consumed since the database restart.
checkpoints_closed	**bigint**	Count of checkpoints closed since the database restart.
last_closed_checkpoint_ts	**bigint**	Timestamp of the last closed checkpoint.
hardened_recovery_lsn	**numeric(38)**	Recovery will start from this LSN.
hardened_root_file_guid	**uniqueidentifier**	GUID of the root file that hardened as a result of the last completed checkpoint.
hardened_root_file_watermark	**bigint**	**Internal Only**. How far it is valid to read the root file up to (this is an internally relevant type only – called BSN).
hardened_truncation_lsn	**numeric(38)**	LSN of the truncation point.
log_bytes_since_last_close	**bigint**	Bytes from last close to the current end of log.
time_since_last_close_in_ms	**bigint**	Time since last close of the checkpoint.

COLUMN NAME	TYPE	DESCRIPTION
current_checkpoint_id	**bigint**	Currently new segments are being assigned to this checkpoint. The checkpoint system is a pipeline. The current checkpoint is the one which segments from the log are being assigned to. Once it's reached a limit, the checkpoint is released by the controller and a new one created as current.
current_checkpoint_segment_count	**bigint**	Count of segments in the current checkpoint.
recovery_lsn_candidate	**bigint**	**Internally Only**. Candidate to be picked as recoverylsn when current_checkpoint_id closes.
outstanding_checkpoint_count	**bigint**	Number of checkpoints in the pipeline waiting to be closed.
closing_checkpoint_id	**bigint**	ID of the closing checkpoint. Serializers are working in parallel, so once they're finished then the checkpoint is a candidate to be closed by close thread. But the close thread can only close one at a time and it must be in order, so the closing checkpoint is the one that the close thread is working on.
recovery_checkpoint_id	**bigint**	ID of the checkpoint to be used in recovery.
recovery_checkpoint_ts	**bigint**	Time stamp of recovery checkpoint.
bootstrap_recovery_lsn	**numeric(38)**	Recovery LSN for the bootstrap.
bootstrap_root_file_guid	**uniqueidentifier**	GUID of the root file for the bootstrap.
internal_error_code	**bigint**	Error seen by any of the controller, serializer, close, and merge threads.
bytes_of_large_data_serialized	**bigint**	The amount of data that was serialized.

SQL Server 2014 (12.x)

The following table describes the columns in `sys.dm_db_xtp_checkpoint_stats`, for **SQL Server 2014 (12.x)**.

COLUMN NAME	TYPE	DESCRIPTION
log_to_process_in_bytes	**bigint**	The number of log bytes between the thread's current log sequence number (LSN) and the end-of-log.

COLUMN NAME	TYPE	DESCRIPTION
total_log_blocks_processed	**bigint**	Total number of log blocks processed since server startup.
total_log_records_processed	**bigint**	Total number of log records processed since server startup.
xtp_log_records_processed	**bigint**	Total number of In-Memory OLTP log records processed since server startup.
total_wait_time_in_ms	**bigint**	Cumulative wait time in ms.
waits_for_io	**bigint**	Number of waits for log IO.
io_wait_time_in_ms	**bigint**	Cumulative time spent waiting on log IO.
waits_for_new_log	**bigint**	Number of waits for new log to be generated.
new_log_wait_time_in_ms	**bigint**	Cumulative time spend waiting on new log.
log_generated_since_last_checkpoint_in_bytes	**bigint**	Amount of log generated since the last In-Memory OLTP checkpoint.
ms_since_last_checkpoint	**bigint**	Amount of time in milliseconds since the last In-Memory OLTP checkpoint.
checkpoint_lsn	**numeric (38)**	The recovery log sequence number (LSN) associated with the last completed In-Memory OLTP checkpoint.
current_lsn	**numeric (38)**	The LSN of the log record that is currently processing.
end_of_log_lsn	**numeric (38)**	The LSN of the end of the log.
task_address	**varbinary(8)**	The address of the SOS_Task. Join to sys.dm_os_tasks to find additional information.

Permissions

Requires `VIEW DATABASE STATE` permission on the server.

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_checkpoint_files (Transact-SQL)

5/4/2018 • 8 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Displays information about checkpoint files, including file size, physical location and the transaction ID.

> **NOTE:** For the current checkpoint that has not closed, the state column of s `ys.dm_db_xtp_checkpoint_files` will be UNDER CONSTRUCTION for new files. A checkpoint closes automatically when there is sufficient transaction log growth since the last checkpoint, or if you issue the `CHECKPOINT` command (CHECKPOINT (Transact-SQL)).

A memory-optimized file group internally uses append-only files to store inserted and deleted rows for in-memory tables. There are two types of files. A data file contains inserted rows while a delta file contains references to deleted rows.

SQL Server 2014 (12.x) is substantially different from more recent versions and is discussed lower in the topic at SQL Server 2014.

For more information, see Creating and Managing Storage for Memory-Optimized Objects.

SQL Server 2016 (13.x) and later

The following table describes the columns for `sys.dm_db_xtp_checkpoint_files`, beginning with **SQL Server 2016 (13.x)**.

COLUMN NAME	TYPE	DESCRIPTION
container_id	**int**	The ID of the container (represented as a file with type FILESTREAM in sys.database_files) that the data or delta file is part of. Joins with file_id in sys.database_files (Transact-SQL).
container_guid	**uniqueidentifier**	GUID of the Container, which the root, data or delta file is part of. Joins with file_guid in the sys.database_files table.
checkpoint_file_id	**uniqueidentifier**	GUID of the checkpoint file.
relative_file_path	**nvarchar(256)**	Path of the file relative to container it is mapped to.
file_type	**smallint**	-1 for FREE 0 for DATA file. 1 for DELTA file. 2 for ROOT file 3 for LARGE DATA file

COLUMN NAME	TYPE	DESCRIPTION
file_type_desc	**nvarchar(60)**	FREE- All files maintained as FREE are available for allocation. Free files can vary in size depending on anticipated needs by the system. The maximum size is 1GB. DATA - Data files contain rows that have been inserted into memory-optimized tables. DELTA - Delta files contain references to rows in data files that have been deleted. ROOT - Root files contain system metadata for memory-optimized and natively compiled objects. LARGE DATA - Large data files contain values inserted in (n)varchar(max) and varbinary(max) columns, as well as the column segments that are part of columnstore indexes on memory-optimized tables.
internal_storage_slot	**int**	The index of the file in the internal storage array. NULL for ROOT or for state other than 1.
checkpoint_pair_file_id	**uniqueidentifier**	Corresponding DATA or DELTA file. NULL for ROOT.
file_size_in_bytes	**bigint**	Size of the file on the disk.
file_size_used_in_bytes	**bigint**	For checkpoint file pairs that are still being populated, this column will be updated after the next checkpoint.
logical_row_count	**bigint**	For Data, number of rows inserted. For Delta, number of rows deleted after accounting for drop table. For Root, NULL.
state	**smallint**	0 – PRECREATED 1 - UNDER CONSTRUCTION 2 - ACTIVE 3 – MERGE TARGET 8 – WAITING FOR LOG TRUNCATION

COLUMN NAME	TYPE	DESCRIPTION
state_desc	**nvarchar(60)**	PRECREATED – A number of checkpoint files are pre-allocated to minimize or eliminate any waits to allocate new files as transactions are being executed. These precreated files can vary in size, depending on the estimated needs of the workload, but they contain no data. This is a storage overhead in databases with a MEMORY_OPTIMIZED_DATA filegroup. UNDER CONSTRUCTION - These checkpoint files are under construction, meaning they are being populated based on the log records generated by the database, and are not yet part of a checkpoint. ACTIVE - These contain the inserted/deleted rows from previous closed checkpoints. They contain the contents of the tables that area read into memory before applying the active part of the transaction log at the database restart. We expect that size of these checkpoint files to be approximately 2x of the in-memory size of memory-optimized tables, assuming the merge operation is keeping up with the transactional workload. MERGE TARGET – The target of merge operations - these checkpoint files store the consolidated data rows from the source files that were identified by the merge policy. Once the merge is installed, the MERGE TARGET transitions into ACTIVE state. WAITING FOR LOG TRUNCATION – Once the merge has been installed and the MERGE TARGET CFP is part of durable checkpoint, the merge source checkpoint files transition into this state. Files in this state are needed for operational correctness of the database with memory-optimized table. For example, to recover from a durable checkpoint to go back in time.
lower_bound_tsn	**bigint**	Lower bound of the transaction in the file; null if state not in (1, 3).
upper_bound_tsn	**bigint**	Upper bound of the transaction in the file; null if state not in (1, 3).
begin_checkpoint_id	**bigint**	ID of the begin checkpoint.
end_checkpoint_id	**bigint**	ID of the end checkpoint.

COLUMN NAME	TYPE	DESCRIPTION
last_updated_checkpoint_id	**bigint**	ID of the last checkpoint that updated this file.
encryption_status	**smallint**	0, 1, 2
encryption_status_desc	**nvarchar(60)**	0 => UNENCRTPTED 1 => ENCRYPTED WITH KEY 1 2 => ENCRYPTED WITH KEY 2. Valid only for active files.

SQL Server 2014 (12.x)

The following table describes the columns for `sys.dm_db_xtp_checkpoint_files`, for **SQL Server 2014 (12.x)**.

COLUMN NAME	TYPE	DESCRIPTION
container_id	**int**	The ID of the container (represented as a file with type FILESTREAM in sys.database_files) that the data or delta file is part of. Joins with file_id in sys.database_files (Transact-SQL).
container_guid	**uniqueidentifier**	The GUID of the container that the data or delta file is part of.
checkpoint_file_id	**GUID**	ID of the data or delta file.
relative_file_path	**nvarchar(256)**	Path to the data or delta file, relative to the location of the container.
file_type	**tinyint**	0 for data file. 1 for delta file. NULL if the state column is set to 7.
file_type_desc	**nvarchar(60)**	The type of file: DATA_FILE, DELTA_FILE, or NULL if the state column is set to 7.
internal_storage_slot	**int**	The index of the file in the internal storage array. NULL if the state column is not 2 or 3.
checkpoint_pair_file_id	**uniqueidentifier**	The corresponding data or delta file.
file_size_in_bytes	**bigint**	Size of the file that is used. NULL if the state column is set to 5, 6, or 7.

COLUMN NAME	TYPE	DESCRIPTION
file_size_used_in_bytes	**bigint**	Used size of the file that is used. NULL if the state column is set to 5, 6, or 7. For checkpoint file pairs that are still being populated, this column will be updated after the next checkpoint.
inserted_row_count	**bigint**	Number of rows in the data file.
deleted_row_count	**bigint**	Number of deleted rows in the delta file.
drop_table_deleted_row_count	**bigint**	The number of rows in the data files affected by a drop table. Applies to data files when the state column equals 1. Shows deleted row counts from dropped table(s). The drop_table_deleted_row_count statistics are compiled after the memory garbage collection of the rows from dropped table(s) is complete and a checkpoint is taken. If you restart SQL Server before the drop tables statistics are reflected in this column, the statistics will be updated as part of recovery. The recovery process does not load rows from dropped tables. Statistics for dropped tables are compiled during the load phase and reported in this column when recovery completes.
state	**int**	0 – PRECREATED 1 – UNDER CONSTRUCTION 2 - ACTIVE 3 – MERGE TARGET 4 – MERGED SOURCE 5 – REQUIRED FOR BACKUP/HA 6 – IN TRANSITION TO TOMBSTONE 7 – TOMBSTONE
state_desc	**nvarchar(60)**	PRECREATED – A small set of data and delta file pairs, also known as checkpoint file pairs (CFPs) are kept pre-allocated to minimize or eliminate any waits to allocate new files as transactions are being executed. These are full sized with data file size of 128MB and delta file size of 8 MB but contain no data. The number of CFPs is computed as the number of logical processors or schedulers (one per core, no maximum) with a minimum of 8. This is a fixed storage overhead in databases

COLUMN NAME	TYPE	DESCRIPTION
		with memory-optimized tables. UNDER CONSTRUCTION – Set of CFPs that store newly inserted and possibly deleted data rows since the last checkpoint. ACTIVE - These contain the inserted and deleted rows from previous closed checkpoints. These CFPs contain all required inserted and deleted rows required before applying the active part of the transaction log at the database restart. The size of these CFPs will be approximately 2 times the in-memory size of memory-optimized tables, assuming the merge operation is current with the transactional workload. MERGE TARGET – The CFP stores the consolidated data rows from the CFP(s) that were identified by the merge policy. Once the merge is installed, the MERGE TARGET transitions into ACTIVE state. MERGED SOURCE – Once the merge operation is installed, the source CFPs are marked as MERGED SOURCE. Note, the merge policy evaluator may identify multiple merges but a CFP can only participate in one merge operation. REQUIRED FOR BACKUP/HA – Once the merge has been installed and the MERGE TARGET CFP is part of durable checkpoint, the merge source CFPs transition into this state. CFPs in this state are needed for operational correctness of the database with memory-optimized table. For example, to recover from a durable checkpoint to go back in time. A CFP can be marked for garbage collection once the log truncation point moves beyond its transaction range. IN TRANSITION TO TOMBSTONE – These CFPs are not needed by the In-Memory OLTP engine and can they can be garbage collected. This state indicates that these CFPs are waiting for the background thread to transition them to the next state, which is TOMBSTONE. TOMBSTONE – These CFPs are waiting to be garbage collected by the filestream garbage collector. (sp_filestream_force_garbage_collection (Transact-SQL))
lower_bound_tsn	**bigint**	The lower bound of transactions contained in the file. Null if the state column is other than 2, 3, or 4.

COLUMN NAME	TYPE	DESCRIPTION
upper_bound_tsn	**bigint**	The upper bound of transactions contained in the file. Null if the state column is other than 2, 3, or 4.
last_backup_page_count	**int**	Logical page count that is determined at last backup. Applies when the state column is set to 2, 3, 4, or 5. NULL if page count not known.
delta_watermark_tsn	**int**	The transaction of the last checkpoint that wrote to this delta file. This is the watermark for the delta file.
last_checkpoint_recovery_lsn	**nvarchar(23)**	Recovery log sequence number of the last checkpoint that still needs the file.
tombstone_operation_lsn	**nvarchar(23)**	The file will be deleted once the tombstone_operation_lsn falls behind the log truncation log sequence number.
logical_deletion_log_block_id	**bigint**	Applies only to state 5.

Permissions

Requires `VIEW DATABASE STATE` permission on the server.

Use Cases

You can estimate the storage used by In-Memory OLTP as follows:

```
-- total storage used by In-Memory OLTP
SELECT SUM (file_size_in_bytes)/(1024*1024) as file_size_in_MB
FROM sys.dm_db_xtp_checkpoint_files
```

To see a breakdown of storage utilization by state and file type run the following query:

```
SELECT state_desc
 , file_type_desc
 , COUNT(*) AS [count]
 , SUM(file_size_in_bytes) / 1024 / 1024 AS [on-disk size MB]
FROM sys.dm_db_xtp_checkpoint_files
GROUP BY state, state_desc, file_type, file_type_desc
ORDER BY state, file_type
```

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_gc_cycle_stats (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Outputs the current state of committed transactions that have deleted one or more rows. The idle garbage collection thread wakes every minute or when the number of committed DML transactions exceeds an internal threshold since the last garbage collection cycle. As part of the garbage collection cycle, it moves the transactions that have committed into one or more queues associated with generations. The transactions that have generated stale versions are grouped in a unit of 16 transactions across 16 generations as follows:

- Generation-0: This stores all transactions that committed earlier than the oldest active transaction. Row versions generated by these transactions are immediately available for garbage collection.
- Generations 1-14: Stores transactions with timestamp greater than the oldest active transaction. The row versions cannot be garbage collected. Each generation can hold up to 16 transactions. A total of 224 (14 * 16) transactions can exist in these generations.
- Generation 15: The remaining transactions with timestamp greater than the oldest active transaction go to generation 15. Similar to generation-0, there is no limit of number of transactions in generation-15.

  When there is memory pressure, the garbage collection thread updates the oldest active transaction hint aggressively, which forces garbage collection.

  For more information, see In-Memory OLTP (In-Memory Optimization).

COLUMN NAME	TYPE	DESCRIPTION
cycle_id	**bigint**	A unique identifier for the garbage collection cycle.
ticks_at_cycle_start	**bigint**	Ticks at the time the cycle started.
ticks_at_cycle_end	**bigint**	Ticks at the time the cycle ended.
base_generation	**bigint**	The current base generation value in the database. This represents the timestamp of the oldest active transaction used to identify transactions for garbage collection. The oldest active transaction id is updated in the increment of 16. For example, if you have transaction ids as 124, 125, 126 ... 139, the value will be 124. When you add another transaction, for example 140, the value will be 140.
xacts_copied_to_local	**bigint**	The number of transactions copied from the transaction pipeline into the database's generation array.
xacts_in_gen_0- xacts_in_gen_15	**bigint**	Number of transactions in each generation.

Permissions

Requires VIEW DATABASE STATE permission on the server.

Usage Scenario

Here is a sample output with a subset of columns, showing 27 generations:

```
cycle_id   ticks_at_cycle_start ticks_at_cycle_end   base_generation  xacts_in_gen_0    xacts_in_gen_1

1          123160509            123160509            1                  0                  0
2          123176822            123176822            1                  0                  1
3          123236826            123236826            1                  0                  1
4          123296829            123296829            1                  0                  1
5          123356832            123356941            129                0                  0
6          123357473            123357473            129                0                  0
7          123417486            123417486            129                0                  0
8          123477489            123477489            129                0                  0
9          123537492            123537492            129                0                  0
10         123597500            123597500            129                0                  0
11         123657504            123657504            129                0                  0
12         123717507            123717507            129                0                  0
13         123777510            123777510            129                0                  0
14         123837513            123837513            129                0                  0
15         123897516            123897516            129                0                  0
16         123957516            123957516            129                0                  0
17         124017516            124017516            129                0                  0
18         124077517            124077517            129                0                  0
19         124137517            124137517            129                0                  0
20         124197518            124197518            129                0                  0
21         124257518            124257518            129                0                  0
22         124317523            124317523            129                0                  0
23         124377526            124377526            129                0                  0
24         124437529            124437529            129                0                  0
25         124497533            124497533            129                0                  0
26         124557536            124557536            129                0                  0
27         124617539            124617539            129                0                  0
```

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_hash_index_stats (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

These statistics are useful for understanding and tuning the bucket counts. It can also be used to detect cases where the index key has many duplicates.

A large average chain length indicates that many rows are hashed to the same bucket. This could happen because:

- If the number of empty buckets is low or the average and maximum chain lengths are similar, it is likely that the total bucket count is too low. This causes many different index keys to hash to the same bucket.
- If the number of empty buckets is high or the maximum chain length is high relative to the average chain length, it is likely that there are many rows with duplicate index key values or there is a skew in the key values. All rows with the same index key value hash to the same bucket, hence there is a long chain length in that bucket.

Long chain lengths can significantly impact the performance of all DML operations on individual rows, including SELECT and INSERT. Short chain lengths along with a high empty bucket count are in indication of a bucket_count that is too high. This decreases the performance of index scans.

> **WARNING**
> **sys.dm_db_xtp_hash_index_stats** scans the entire table. So, if there are large tables in your database, **sys.dm_db_xtp_hash_index_stats** may take a long time run.

For more information, see Hash Indexes for Memory-Optimized Tables.

COLUMN NAME	TYPE	DESCRIPTION
object_id	**int**	The object ID of parent table.
xtp_object_id	**bigint**	ID of the memory-optimized table.
index_id	**int**	The index ID.
total_bucket_count	**bigint**	The total number of hash buckets in the index.
empty_bucket_count	**bigint**	The number of empty hash buckets in the index.
avg_chain_length	**bigint**	The average length of the row chains over all the hash buckets in the index.
max_chain_length	**bigint**	The maximum length of the row chains in the hash buckets.

COLUMN NAME	TYPE	DESCRIPTION
xtp_object_id	**bigint**	The in-memory OLTP object ID that corresponds to the memory-optimized table.

Permissions

Requires VIEW DATABASE STATE permission on the server.

Examples

A. Troubleshooting hash index bucket count

The following query can be used to troubleshoot the hash index bucket count of an existing table. The query returns statistics about percentage of empty buckets and chain length for all hash indexes on user tables.

```
SELECT
  QUOTENAME(SCHEMA_NAME(t.schema_id)) + N'.' + QUOTENAME(OBJECT_NAME(h.object_id)) as [table],
  i.name                   as [index],
  h.total_bucket_count,
  h.empty_bucket_count,
  FLOOR((
    CAST(h.empty_bucket_count as float) /
      h.total_bucket_count) * 100)
                           as [empty_bucket_percent],
  h.avg_chain_length,
  h.max_chain_length
FROM sys.dm_db_xtp_hash_index_stats as h
INNER JOIN sys.indexes as i
          ON h.object_id = i.object_id
         AND h.index_id  = i.index_id
  INNER JOIN sys.memory_optimized_tables_internal_attributes ia ON h.xtp_object_id=ia.xtp_object_id
  INNER JOIN sys.tables t on h.object_id=t.object_id
WHERE ia.type=1
ORDER BY [table], [index];
```

For details on how to interpret the results of this query, see Troubleshooting Hash Indexes for Memory-Optimized Tables .

B. Hash index statistics for internal tables

Certain features use internal tables that leverage hash indexes, for example columnstore indexes on memory-optimized tables. The following query returns stats for hash indexes on internal tables that are linked to user tables.

```
SELECT
  QUOTENAME(SCHEMA_NAME(t.schema_id)) + N'.' + QUOTENAME(OBJECT_NAME(h.object_id)) as [user_table],
  ia.type_desc as [internal_table_type],
  i.name                    as [index],
  h.total_bucket_count,
  h.empty_bucket_count,
  h.avg_chain_length,
  h.max_chain_length
FROM sys.dm_db_xtp_hash_index_stats as h
INNER JOIN sys.indexes as i
          ON h.object_id = i.object_id
         AND h.index_id  = i.index_id
  INNER JOIN sys.memory_optimized_tables_internal_attributes ia ON h.xtp_object_id=ia.xtp_object_id
  INNER JOIN sys.tables t on h.object_id=t.object_id
WHERE ia.type!=1
ORDER BY [user_table], [internal_table_type], [index];
```

Note that the BUCKET_COUNT of index on internal tables cannot be changed, thus the output of this query should be considered informative only. No action is required.

This query is not expected to return any rows unless you are using a feature that leverages hash indexes on internal tables. The following memory-optimized table contains a columnstore index. After creating this table, you will see hash indexes on internal tables.

```
CREATE TABLE dbo.table_columnstore
(
  c1 INT NOT NULL PRIMARY KEY NONCLUSTERED,
  INDEX ix_columnstore CLUSTERED COLUMNSTORE
) WITH (MEMORY_OPTIMIZED=ON)
```

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_index_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Contains statistics collected since the last database restart.

For more information, see In-Memory OLTP (In-Memory Optimization) and Guidelines for Using Indexes on Memory-Optimized Tables.

COLUMN NAME	DATA TYPE	DESCRIPTION
object_id	**bigint**	ID of the object to which this index belongs.
xtp_object_id	**bigint**	Internal ID corresponding to the current version of the object. Note: Applies to SQL Server 2016 (13.x).
index_id	**bigint**	ID of the index. The index_id is unique only within the object.
scans_started	**bigint**	Number of In-Memory OLTP index scans performed. Every select, insert, update, or delete requires an index scan.
scans_retries	**bigint**	Number of index scans that needed to be retried,
rows_returned	**bigint**	Cumulative number of rows returned since the table was created or the start of SQL Server.
rows_touched	**bigint**	Cumulative number of rows accessed since the table was created or the start of SQL Server.
rows_expiring	**bigint**	Internal use only.
rows_expired	**bigint**	Internal use only.
rows_expired_removed	**bigint**	Internal use only.
phantom_scans_started	**bigint**	Internal use only.
phatom_scans_retries	**bigint**	Internal use only.
phantom_rows_touched	**bigint**	Internal use only.

COLUMN NAME	DATA TYPE	DESCRIPTION
phantom_expiring_rows_encountered	**bigint**	Internal use only.
phantom_expired_rows_encountered	**bigint**	Internal use only.
phantom_expired_removed_rows_encountered	**bigint**	Internal use only.
phantom_expired_rows_removed	**bigint**	Internal use only.
object_address	**varbinary(8)**	Internal use only.

Permissions

Requires VIEW DATABASE STATE permission on the current database.

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_memory_consumers (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Reports the database-level memory consumers in the In-Memory OLTP database engine. The view returns a row for each memory consumer that the database engine uses. Use this DMV to see how the memory is distributed across different internal objects.

For more information, see In-Memory OLTP (In-Memory Optimization).

COLUMN NAME	DATA TYPE	DESCRIPTION
memory_consumer_id	**bigint**	ID (internal) of the memory consumer.
memory_consumer_type	**int**	The type of memory consumer: 0=Aggregation. (Aggregates memory usage of two or more consumers. It should not be displayed.) 2=VARHEAP (Tracks memory consumption for a variable-length heap.) 3=HASH (Tracks memory consumption for an index.) 5=DB page pool (Tracks memory consumption for a database page pool used for runtime operations. For example, table variables and some serializable scans. There is only one memory consumer of this type per database.)
memory_consumer_type_desc	**nvarchar(64)**	Type of memory consumer: VARHEAP, HASH, or PGPOOL. 0 – (It should not be displayed.) 2 - VARHEAP 3 - HASH 5 - PGPOOL

COLUMN NAME	DATA TYPE	DESCRIPTION
memory_consumer_desc	**nvarchar(64)**	Description of the memory consumer instance: VARHEAP: Database heap. Used to allocate user data for a database (rows). Database System heap. Used to allocate database data that will be included in memory dumps and do not include user data. Range index heap. Private heap used by range index to allocate BW pages. HASH: No description since the object_id indicates the table and the index_id the hash index itself. PGPOOL: For the database there is only one page pool Database 64K page pool.
object_id	**bigint**	The object ID to which the allocated memory is attributed. A negative value for system objects.
xtp_object_id	**bigint**	The object ID for the memory-optimized table.
index_id	**int**	The index ID of the consumer (if any). NULL for base tables.
allocated_bytes	**bigint**	Number of bytes reserved for this consumer.
used_bytes	**bigint**	Bytes used by this consumer. Applies only to varheap.
allocation_count	**int**	Number of allocations.
partition_count	**int**	Internal use only.
sizeclass_count	**int**	Internal use only.
min_sizeclass	**int**	Internal use only.
max_sizeclass	**int**	Internal use only.
memory_consumer_address	**varbinary**	Internal address of the consumer. For internal use only.
xtp_object_id	**bigint**	The in-memory OLTP object ID that corresponds to the memory-optimized table.

Remarks

In the output, the allocators at database levels refer to user tables, indexes, and system tables. VARHEAP with object_id = NULL refers to memory allocated to tables with variable length columns.

Permissions

All rows are returned if you have VIEW DATABASE STATE permission on the current database. Otherwise, an empty rowset is returned.

If you do not have VIEW DATABASE permission, all columns will be returned for rows in tables that you have SELECT permission on.

System tables are returned only for users with VIEW DATABASE STATE permission.

General Remarks

When a memory-optimized table has a columnstore index, the system uses some internal tables, which consume some memory, to track data for the columnstore index. For details about these internal tables and sample queries showing their memory consumption see sys.memory_optimized_tables_internal_attributes (Transact-SQL).

Examples

```
-- memory consumers (database level)
SELECT OBJECT_NAME(object_id), *
FROM sys.dm_db_xtp_memory_consumers;
```

User Scenario

```
-- memory consumers (database level)

select  convert(char(10), object_name(object_id)) as Name,
convert(char(10),memory_consumer_type_desc ) as memory_consumer_type_desc, object_id,index_id,
allocated_bytes,  used_bytes
from sys.dm_db_xtp_memory_consumers
```

Here is the output with a subset of columns. The allocators at database levels refer to user tables, indexes, and system tables. The VARHEAP with object_id = NULL (last row) refers to memory allocated to data rows of the tables (in the example here, it is t1). The allocated bytes, when converted to MB, is 1340MB.

```
Name       memory_consumer_type_desc object_id   index_id    allocated_bytes      used_bytes
---------- ------------------------- ----------- ----------- -------------------- --------------------
t3         HASH                      629577281   2           8388608              8388608
t2         HASH                      597577167   2           8388608              8388608
t1         HASH                      565577053   2           1048576              1048576
NULL       HASH                      -6          1           2048                 2048
NULL       VARHEAP                   -6          NULL        0                    0
NULL       HASH                      -5          3           8192                 8192
NULL       HASH                      -5          2           8192                 8192
NULL       HASH                      -5          1           8192                 8192
NULL       HASH                      -4          1           2048                 2048
NULL       VARHEAP                   -4          NULL        0                    0
NULL       HASH                      -3          1           2048                 2048
NULL       HASH                      -2          2           8192                 8192
NULL       HASH                      -2          1           8192                 8192
NULL       VARHEAP                   -2          NULL        196608               26496
NULL       HASH                      0           1           2048                 2048
NULL       PGPOOL                    0           NULL        0                    0
NULL       VARHEAP                   NULL        NULL        1405943808           1231220560

(17 row(s) affected)
```

The total memory allocated and used from this DMV is same as the object level in sys.dm_db_xtp_table_memory_stats (Transact-SQL).

```
select  sum(allocated_bytes)/(1024*1024) as total_allocated_MB,
        sum(used_bytes)/(1024*1024) as total_used_MB
from sys.dm_db_xtp_memory_consumers

total_allocated_MB   total_used_MB
-------------------- --------------------
1358                 1191
```

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_merge_requests (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Tracks database merge requests. The merge request may have been generated by SQL Server or the request could have been made by a user with sys.sp_xtp_merge_checkpoint_files (Transact-SQL).

> **NOTE**
>
> This dynamic management view (DMV), sys.dm_db_xtp_merge_requests, exists until Microsoft SQL Server 2014.
>
> But starting with SQL Server 2016 this DMV no longer applies.

Columns in the report

COLUMN NAME	DATA TYPE	DESCRIPTION
request_state	tinyint	Status of the merge request: 0 = requested 1 = pending 2 = installed 3 = abandoned
request_state_desc	nvarchar(60)	Meanings for the current state of the request: Requested - a merge request exists. Pending - the merge is being processing. Installed - the merge is complete. Abandoned - the merge could not complete, perhaps due to lack of storage.
destination_file_id	GUID	The unique identifier of the destination file for the merge of the Source files.
lower_bound_tsn	bigint	The minimum timestamp for the target merge file. The lowest transaction timestamp of all the source files to be merged.
upper_bound_tsn	bigint	The maximum timestamp for the target merge file. The highest transaction timestamp of all the source files to be merged.

COLUMN NAME	DATA TYPE	DESCRIPTION
collection_tsn	bigint	The timestamp at which the current row can be collected. A row in the Installed state is removed when checkpoint_tsn is greater than collection_tsn. A row in the Abandoned state is removed when checkpoint_tsn is less than collection_tsn.
checkpoint_tsn	bigint	The time that the checkpoint started. Any deletes done by transactions with a timestamp lower than this are accounted for in the new data file. The remaining deletes are moved to the target delta file.
sourcenumber_file_id	GUID	Up to 16 internal file ids that uniquely identify the source files in the merge.

Permissions

Requires VIEW DATABASE STATE permission on the current database.

See also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_nonclustered_index_stats (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

sys.dm_db_xtp_nonclustered_index_stats includes statistics about operations on nonclustered indexes in memory-optimized tables. sys.dm_db_xtp_nonclustered_index_stats contains one row for each nonclustered index on a memory-optimized table in the current database.

The statistics reflected in sys.dm_db_xtp_nonclustered_index_stats are collected when the in-memory index structure is created. In-memory index structures are recreated on database restart.

Use sys.dm_db_xtp_nonclustered_index_stats to understand and monitor index activity during DML operations and when a database is brought online. When a database with a memory-optimized table is restarted, the index is built by inserting one row at a time into memory. The count of page splits, merges, and consolidation can help you understand the work done to build the index when a database is brought online. You can also look at these counts before and after a series of DML operations.

Large numbers of retries are indicative of concurrency issues; call Microsoft Support.

For more information about memory-optimized, nonclustered indexes, see SQL Server In-Memory OLTP Internals Overview, page 17.

COLUMN NAME	DATA TYPE	DESCRIPTION
object_id	**int**	ID of the object.
xtp_object_id	**bigint**	ID of the memory-optimized table.
index_id	**int**	ID of the index.
delta_pages	**bigint**	The total number of delta pages for this index in the tree.
internal_pages	**bigint**	For internal use. The total number of internal pages for this index in the tree.
leaf_pages	**bigint**	The total number of leaf pages for this index in the tree.
outstanding_retired_nodes	**bigint**	For internal use. The total number of nodes for this index in the internal structures.
page_update_count	**bigint**	Cumulative number of operations updating a page in the index.
page_update_retry_count	**bigint**	Cumulative number of retries of an operation updating page in the index.

COLUMN NAME	DATA TYPE	DESCRIPTION
page_consolidation_count	**bigint**	Cumulative number of page consolidations in the index.
page_consolidation_retry_count	**bigint**	Cumulative number of retries of page consolidation operations.
page_split_count	**bigint**	Cumulative number of page split operations in the index.
page_split_retry_count	**bigint**	Cumulative number of retries of page split operations.
key_split_count	**bigint**	Cumulative number of key splits in the index.
key_split_retry_count	**bigint**	Cumulative number of retries of key split operations.
page_merge_count	**bigint**	Cumulative number of page merge operations in the index.
page_merge_retry_count	**bigint**	Cumulative number of retries of page merge operations.
key_merge_count	**bigint**	Cumulative number of key merge operations in the index.
key_merge_retry_count	**bigint**	Cumulative number of retries of key merge operations.

Permissions

Requires VIEW DATABASE STATE permission on the current database.

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_object_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Reports the number rows affected by operations on each of the In-Memory OLTP objects since the last database restart. Statistics are updated when the operation executes, regardless of whether the transaction commits or was rolled back.

sys.dm_db_xtp_object_stats can help you identify which memory-optimized tables are changing the most. You may decide to remove unused or rarely used indexes on the table, as each index affects performance. If there are hash indexes, you should periodically re-evaluate the bucket-count. For more information, see Determining the Correct Bucket Count for Hash Indexes.

sys.dm_db_xtp_object_stats can help you identify which memory-optimized tables incur write-write conflicts, which can affect the performance of your application. For example, if you have transaction retry logic, the same statement may need to be executed more than once. Also, you can use this information to identify the tables (and therefore business logic) that require write-write error handling.

The view contains a row for each memory optimized table in the database.

For more information, see In-Memory OLTP (In-Memory Optimization).

COLUMN NAME	DATA TYPE	DESCRIPTION
object_id	**bigint**	The ID of the object.
row_insert_attempts	**bigint**	The number of rows inserted into the table since the last database restart by both committed and aborted transactions.
row_update_attempts	**bigint**	The number of rows updated in the table since the last database restart by both committed and aborted transactions.
row_delete_attempts	**bigint**	The number of rows deleted from the table since the last database restart by both committed and aborted transactions.
write_conflicts	**bigint**	The number of write conflicts that occurred since the last database restart.
unique_constraint_violations	**bigint**	The number of unique constraint violations that have occurred since the last database restart.
object_address	**varbinary(8)**	Internal use only.

Permissions

Requires VIEW DATABASE STATE permission on the current database.

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_table_memory_stats (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns memory usage statistics for each In-Memory OLTP table (user and system) in the current database. The system tables have negative object IDs and are used to store run-time information for the In-Memory OLTP engine. Unlike user objects, system tables are internal and only exist in-memory, therefore, they are not visible through catalog views. System tables are used to store information such as meta-data for all data/delta files in storage, merge requests, watermarks for delta files to filter rows, dropped tables, and relevant information for recovery and backups. Given that the In-Memory OLTP engine can have up to 8,192 data and delta file pairs, for large in-memory databases, the memory taken by system tables can be a few megabytes.

For more information, see In-Memory OLTP (In-Memory Optimization).

COLUMN NAME	DATA TYPE	DESCRIPTION
object_id	**int**	The object ID of the table. NULL for In-Memory OLTP system tables.
memory_allocated_for_table_kb	**bigint**	Memory allocated for this table.
memory_used_by_table_kb	**bigint**	Memory used by table, including row versions.
memory_allocated_for_indexes_kb	**bigint**	Memory allocated for indexes on this table.
memory_used_by_indexes_kb	**bigint**	Memory consumed for indexes on this table.

Permissions

All rows are returned if you have VIEW DATABASE STATE permission on the current database. Otherwise, an empty rowset is returned.

If you do not have VIEW DATABASE permission, all columns will be returned for rows in tables that you have SELECT permission on.

System tables are returned only for users with VIEW DATABASE STATE permission.

Examples

You can query the following DMV to get the memory allocated for the tables and indexes within the database:

```
-- finding memory for objects
SELECT OBJECT_NAME(object_id), *
FROM sys.dm_db_xtp_table_memory_stats;
```

To find memory for all objects within the database:

```
SELECT SUM( memory_allocated_for_indexes_kb + memory_allocated_for_table_kb) AS
 memoryallocated_objects_in_kb
FROM sys.dm_db_xtp_table_memory_stats;
```

User Scenario

First create the following tables in a database called HkDb1.

```
-- set max server memory to 4 GB
EXEC sp_configure 'max server memory (MB)', 4048
go

RECONFIGURE
go

-- create a resource pool for database with memory-optimized objects
CREATE RESOURCE POOL PoolHkDb1 WITH (MAX_MEMORY_PERCENT = 50);
ALTER RESOURCE GOVERNOR RECONFIGURE;
go

--bind the pool to the database
EXEC sp_xtp_bind_db_resource_pool 'HkDb1', 'PoolHkdb1'
go

-- take database offline/online to associate the pool
use master
go

alter database HkDb1 set offline
go
alter database HkDb1 set online
go

USE HkDb1
go

CREATE TABLE dbo.t1 (
       c1 int NOT NULL,
       c2 char(40) NOT NULL,
       c3 char(8000) NOT NULL,

       CONSTRAINT [pk_t1_c1] PRIMARY KEY NONCLUSTERED HASH (c1) WITH (BUCKET_COUNT = 100000)
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA)
go

CREATE TABLE dbo.t2 (
       c1 int NOT NULL,
       c2 char(40) NOT NULL,
       c3 char(8000) NOT NULL,

       CONSTRAINT [pk_t2_c1] PRIMARY KEY NONCLUSTERED HASH (c1) WITH (BUCKET_COUNT = 100000)
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA)
go

CREATE TABLE dbo.t3 (
       c1 int NOT NULL,
       c2 char(40) NOT NULL,
       c3 char(8000) NOT NULL,

       CONSTRAINT [pk_t3_c1] PRIMARY KEY NONCLUSTERED HASH (c1) WITH (BUCKET_COUNT = 1000000)
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA)
go

-- load 150K rows
declare @i int = 0
while (@i <= 150000)
begin
       insert t1 values (@i, 'a', replicate ('b', 8000))
       set @i += 1;
end
go
```

When data is loaded into a table, you can see user defined tables and how much storage it is using. For example, each row of a table could be approximately 8070 bytes (allocation size is 8K (8192 bytes)). You can see indexes per table and how much storage the index uses. For example, 1MB is 100K entries rounded to the next power of 2 (2**17) = 131072 of 8 bytes each. A table may not have an index, in which case it will show memory allocation for the index. Other rows may represent system tables

```
select convert(char(10), object_name(object_id)) as Name,*
from sys.dm_db_xtp_table_memory_stats
```

Here is the output, in two parts:

```
Name       object_id   memory_allocated_for_table_kb memory_used_by_table_kb
---------- ----------- ----------------------------- -----------------------
t3         629577281   0                             0
t1         565577053   1372928                       1202351
t2         597577167   0                             0
NULL       -6          0                             0
NULL       -5          0                             0
NULL       -4          0                             0
NULL       -3          0                             0
NULL       -2          192                           25

memory_allocated_for_indexes_kb memory_used_by_indexes_kb
------------------------------- -------------------------
8192                            8192
1024                            1024
8192                            8192
2                               2
24                              24
2                               2
2                               2
16                              16
```

The output of,

```
select  sum(allocated_bytes)/(1024*1024) as total_allocated_MB,
       sum(used_bytes)/(1024*1024) as total_used_MB
from sys.dm_db_xtp_memory_consumers
```

is,

```
total_allocated_MB   total_used_MB
-------------------- --------------------
1357                 1191
```

Next, let's look at the output from the resource pool. Note, that memory used from the pool is 1356 MB

```
select pool_id,convert(char(10), name) as Name, min_memory_percent, max_memory_percent,
   max_memory_kb/1024 as max_memory_mb
from sys.dm_resource_governor_resource_pools

select used_memory_kb/1024 as used_memory_mb ,target_memory_kb/1024 as target_memory_mb
from sys.dm_resource_governor_resource_pools
```

here is the output:

```
pool_id     Name       min_memory_percent max_memory_percent max_memory_mb
----------- ---------- ------------------ ------------------ --------------------
1           internal   0                  100                3845
2           default    0                  100                3845
259         PoolHkDb1  0                  100                3845

used_memory_mb       target_memory_mb
-------------------- --------------------
125                  3845
32                   3845
1356                 3845
```

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_db_xtp_transactions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Reports the active transactions in the In-Memory OLTP database engine.

For more information, see In-Memory OLTP (In-Memory Optimization).

COLUMN NAME	DATA TYPE	DESCRIPTION
xtp_transaction_id	**bigint**	Internal ID for this transaction in the XTP transaction manager.
transaction_id	**bigint**	The transaction ID. Joins with the transaction ID in other transaction-related DMVs, such as sys.dm_tran_active_transactions. 0 for XTP-only transactions, such as transactions started by natively compiled stored procedures.
session_id	**smallint**	The session identifier of the session that is executing this transaction. Joins with sys.dm_exec_sessions.
begin_tsn	**bigint**	Begin transaction serial number of the transaction.
end_tsn	**bigint**	End transaction serial number of the transaction.
state	**int**	The state of the transaction: 0=ACTIVE 1=COMMITTED 2=ABORTED 3=VALIDATING
state_desc	**nvarchar**	The description of the transaction state.

COLUMN NAME	DATA TYPE	DESCRIPTION
result	**int**	The result of this transaction. The following are the possible values. 0 - IN PROGRESS 1 - SUCCESS 2 - ERROR 3 - COMMIT DEPENDENCY 4 - VALIDATION FAILED (RR) 5 - VALIDATION FAILED (SR) 6 - ROLLBACK
result_desc	**nvarchar**	The result of this transaction. The following are the possible values. IN PROGRESS SUCCESS ERROR COMMIT DEPENDENCY VALIDATION FAILED (RR) VALIDATION FAILED (SR) ROLLBACK
last_error	**int**	Internal use only
is_speculative	**bit**	Internal use only
is_prepared	**bit**	Internal use only
is_delayed_durability	**bit**	Internal use only
memory_address	**varbinary**	Internal use only
database_address	**varbinary**	Internal use only
thread_id	**int**	Internal use only
read_set_row_count	**int**	Internal use only
write_set_row_count	**int**	Internal use only
scan_set_count	**int**	Internal use only
savepoint_garbage_count	**int**	Internal use only

COLUMN NAME	DATA TYPE	DESCRIPTION
log_bytes_required	**bigint**	Internal use only
count_of_allocations	**int**	Internal use only
allocated_bytes	**int**	Internal use only
reserved_bytes	**int**	Internal use only
commit_dependency_count	**int**	Internal use only
commit_dependency_total_attempt_count	**int**	Internal use only
scan_area	**int**	Internal use only
scan_area_desc	**nvarchar**	Internal use only
scan_location	**int**	Internal use only.
dependent_1_address	**varbinary(8)**	Internal use only
dependent_2_address	**varbinary(8)**	Internal use only
dependent_3_address	**varbinary(8)**	Internal use only
dependent_4_address	**varbinary(8)**	Internal use only
dependent_5_address	**varbinary(8)**	Internal use only
dependent_6_address	**varbinary(8)**	Internal use only
dependent_7_address	**varbinary(8)**	Internal use only
dependent_8_address	**varbinary(8)**	Internal use only

Permissions

Requires VIEW DATABASE STATE permission on the server.

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_xtp_gc_queue_stats (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Outputs information about each garbage collection worker queue on the server, and various statistics about each. There is one queue per logical CPU.

The main garbage collection thread (the Idle thread) tracks updated, deleted, and inserted rows for all transactions completed since the last invocation of the main garbage collection thread. When the garbage collection thread wakes, it determines if the timestamp of the oldest active transaction has changed. If the oldest active transaction has changed, then the idle thread enqueues work items (in chunks of 16 rows) for transactions whose write sets are no longer needed. For example, if you delete 1,024 rows, you will eventually see 64 garbage collection work items queued, each containing 16 deleted rows. After a user transaction commits, it selects all enqueued items on its scheduler. If there are no enqueued items on its scheduler, the user transaction will search on any queue in the current NUMA node.

You can determine if garbage collection is freeing memory for deleted rows by executing sys.dm_xtp_gc_queue_stats to see if the enqueued work is being processed. If entries in the current_queue_depth are not being processed or if no new work items are being added to the current_queue_depth, this is an indication that garbage collection is not freeing memory. For example, garbage collection can't be done if there is a long running transaction.

For more information, see In-Memory OLTP (In-Memory Optimization).

COLUMN NAME	TYPE	DESCRIPTION
queue_id	**int**	The unique identifier of the queue.
total_enqueues	**bigint**	The total number of garbage collection work items enqueued to this queue since the server started.
total_dequeues	**bigint**	The total number of garbage collection work items dequeued from this queue since the server started.
current_queue_depth	**bigint**	The current number of garbage collection work items present on this queue. This item may imply one or more to be garbage collected.
maximum_queue_depth	**bigint**	The maximum depth this queue has seen.
last_service_ticks	**bigint**	CPU ticks at the time the queue was last serviced.

Permissions

Requires VIEW SERVER STATE permission.

User Scenario

This output shows that SQL Server is either running on 4 cores or SQL Server instance has been affinitized to 4 cores:

This output shows that there are no work items in the queues to process. For queue 0, the total work items de-queued since SQL Startup are 15625 and the max queue depth has been 215625.

```
queue_id total_enqueues total_dequeues current_queue_depth  maximum_queue_depth  last_service_ticks
---------------------------------------------------------------------------------------------------
0        15625                15625    0                    15625                1233573168347
1        15625                15625    0                    15625                1234123295566
2        15625                15625    0                    15625                1233569418146
3        15625                15625    0                    15625                1233571605761
```

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_xtp_gc_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Provides information (the overall statistics) about the current behavior of the In-Memory OLTP garbage-collection process.

Rows are garbage collected as part of regular transaction processing, or by the main garbage collection thread, which is referred to as the idle worker. When a user transaction commits, it dequeues one work item from the garbage collection queue (sys.dm_xtp_gc_queue_stats (Transact-SQL)). Any rows that could be garbage collected but were not accessed by main user transaction are garbage collected by the idle worker, as part of the dusty corner scan (a scan for areas of the index that are less accessed).

For more information, see In-Memory OLTP (In-Memory Optimization).

COLUMN NAME	TYPE	DESCRIPTION
rows_examined	**bigint**	The number of rows examined by the garbage collection subsystem since the server was started.
rows_no_sweep_needed	**bigint**	The number of rows that were removed without a dusty corner scan.
rows_first_in_bucket	**bigint**	The number of rows examined by garbage collection that were the first row in the hash bucket.
rows_first_in_bucket_removed	**bigint**	The number of rows examined by garbage collection that were the first row in the hash bucket that have been removed.
rows_marked_for_unlink	**bigint**	The number of rows examined by garbage collection that were already marked as unlinked in their indexes with ref count =0.
parallel_assist_count	**bigint**	The number of rows processed by user transactions.
idle_worker_count	**bigint**	The number of garbage rows processed by the idle worker.
sweep_scans_started	**bigint**	The number of dusty corner scans performed by garbage collection subsystem.
sweep_scans_retries	**bigint**	The number of dusty corner scans performed by the garbage collection subsystem.

COLUMN NAME	TYPE	DESCRIPTION
sweep_rows_touched	**bigint**	Rows read by dusty corner processing.
sweep_rows_expiring	**bigint**	Expiring rows read by dusty corner processing.
sweep_rows_expired	**bigint**	Expired rows read by dusty corner processing.
sweep_rows_expired_removed	**bigint**	Expired rows removed by dusty corner processing.

Permissions

Requires VIEW SERVER STATE permission on the instance.

Usage Scenario

The following is sample output:

```
rows_examined        rows_no_sweep_needed rows_first_in_bucket rows_first_in_bucket_removed
280085               209512               69905
rows_first_in_bucket_removed rows_marked_for_unlink parallel_assist_count idle_worker_count
69905                        0                      8953

idle_worker_count    sweep_scans_started  sweep_scan_retries   sweep_rows_touched
10306473             670                  0                    1343

sweep_rows_expiring  sweep_rows_expired   sweep_rows_expired_removed
              0                673673
```

See Also

[Memory-Optimized Table Dynamic Management Views (Transact-SQL)]

sys.dm_xtp_system_memory_consumers (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Reports system level memory consumers for In-Memory OLTP. The memory for these consumers come either from the default pool (when the allocation is in the context of a user thread) or from internal pool (if the allocation is in the context of a system thread).

```
-- system memory consumers @ instance
select * from sys.dm_xtp_system_memory_consumers
```

For more information, see In-Memory OLTP (In-Memory Optimization).

COLUMN NAME	TYPE	DESCRIPTION
memory_consumer_id	**bigint**	Internal ID for memory consumer.
memory_consumer_type	**int**	An integer that represents the type of the memory consumer with one of the following values: 0 – It should not be displayed. Aggregates memory usage of two or more consumers. 1 – LOOKASIDE: Tracks memory consumption for a system lookaside. 2 - VARHEAP: Tracks memory consumption for a variable-length heap. 4 - IO page pool: Tracks memory consumption for a system page pool used for IO operations.
memory_consumer_type_desc	**nvarchar(16)**	The description of the type of memory consumer: 0 – It should not be displayed. 1 – LOOKASIDE 2 - VARHEAP 4 - PGPOOL

COLUMN NAME	TYPE	DESCRIPTION
memory_consumer_desc	**nvarchar(64)**	Description of the memory consumer instance: VARHEAP: System heap. General purpose. Currently only used to allocate garbage collection work items. -OR- Lookaside heap. Used by looksides when the number of items contained in the lookaside list reaches a predetermined cap (usually around 5,000 items). PGPOOL: For IO system pools there are three different sizes System 4K page pool, System 64K page pool, and System 256K page pool.
lookaside_id	**bigint**	The ID of the thread-local, lookaside memory provider.
pagepool_id	**bigint**	The ID of the thread-local, page pool memory provider.
allocated_bytes	**bigint**	Number of bytes reserved for this consumer.
used_bytes	**bigint**	Bytes used by this consumer. Applies only to varheap memory consumers.
allocation_count	**int**	Number of allocations.
partition_count	**int**	Internal use only.
sizeclass_count	**int**	Internal use only.
min_sizeclass	**int**	Internal use only.
max_sizeclass	**int**	Internal use only.
memory_consumer_address	**varbinary**	Internal address of the consumer.

Permissions

Requires VIEW SERVER STATE permissions on the server.

User Scenario

```
-- system memory consumers @ instance
selectmemory_consumer_type_desc,
allocated_bytes/1024 as allocated_bytes_kb,
used_bytes/1024 as used_bytes_kb, allocation_count
from sys.dm_xtp_system_memory_consumers
```

The output shows all memory consumers at system level. For example, there are consumers for transaction look aside.

```
memory_consumer_type_name           memory_consumer_desc                              allocated_bytes_kb
used_bytes_kb        allocation_count
-------------------------------          ---------------------                          -------------------  -
-------------        ----------------
VARHEAP                                  Lookaside heap                                 0                    0
0
VARHEAP                                  System heap                                    768                  0
2
LOOKASIDE                                GC transaction map entry                       64
64                   910
LOOKASIDE                                Redo transaction map entry                     128
128                  1260
LOOKASIDE                                Recovery table cache entry                     448
448                  8192
LOOKASIDE                                Transaction recent rows                        3264
3264                 4444
LOOKASIDE                                Range cursor                                   0                    0
0
LOOKASIDE                                Hash cursor                                    3200
3200                 11070
LOOKASIDE                                Transaction save-point set entry               0                    0
0
LOOKASIDE                                Transaction partially-inserted rows set        704
704                  1287
LOOKASIDE                                Transaction constraint set                     576
576                  1940
LOOKASIDE                                Transaction save-point set                     0                    0
0
LOOKASIDE                                Transaction write set                          704
704                  672
LOOKASIDE                                Transaction scan set                           320
320                  156
LOOKASIDE                                Transaction read set                           704
704                  343
LOOKASIDE                                Transaction                                    4288
4288                 1459
PGPOOL                                   System 256K page pool                          5120
5120                 20
PGPOOL                                   System 64K page pool                           0                    0
0
PGPOOL                                   System 4K page pool                            24
24                   6
```

To see the total memory consumed by system allocators:

```
select sum(allocated_bytes)/(1024*1024) as total_allocated_MB, sum(used_bytes)/(1024*1024) as total_used_MB
from sys.dm_xtp_system_memory_consumers

total_allocated_MB   total_used_MB
-------------------- --------------------
2                    2
```

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

sys.dm_xtp_transaction_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Reports statistics about transactions that have run since the server started.

For more information, see In-Memory OLTP (In-Memory Optimization).

COLUMN NAME	DATA TYPE	DESCRIPTION
total_count	**bigint**	The total number of transactions that have run in the In-Memory OLTP database engine.
read_only_count	**bigint**	The number of read-only transactions.
total_aborts	**bigint**	Total number of transactions that were aborted, either through user or system abort.
user_aborts	**bigint**	Number of aborts initiated by the system. For example, because of write conflicts, validation failures, or dependency failures.
validation_failures	**bigint**	The number of times a transaction has aborted due to a validation failure.
dependencies_taken	**bigint**	Internal use only.
dependencies_failed	**bigint**	The number of times a transaction aborts because a transaction on which it was dependent aborts.
savepoint_create	**bigint**	The number of savepoints created. A new savepoint is created for every atomic block.
savepoint_rollbacks	**bigint**	The number of rollbacks to a previous savepoint.
savepoint_refreshes	**bigint**	Internal use only.
log_bytes_written	**bigint**	Total number of bytes written to the In-Memory OLTP log records.
log_IO_count	**bigint**	Total number of transactions that require log IO. Only considers transactions on durable tables.
phantom_scans_started	**bigint**	Internal use only.

COLUMN NAME	DATA TYPE	DESCRIPTION
phatom_scans_retries	**bigint**	Internal use only.
phantom_rows_touched	**bigint**	Internal use only.
phantom_rows_expiring	**bigint**	Internal use only.
phantom_rows_expired	**bigint**	Internal use only.
phantom_rows_expired_removed	**bigint**	Internal use only.
scans_started	**bigint**	Internal use only.
scans_retried	**bigint**	Internal use only.
rows_returned	**bigint**	Internal use only.
rows_touched	**bigint**	Internal use only.
rows_expiring	**bigint**	Internal use only.
rows_expired	**bigint**	Internal use only.
rows_expired_removed	**bigint**	Internal use only.
rows_inserted	**bigint**	Internal use only.
rows_updated	**bigint**	Internal use only.
rows_deleted	**bigint**	Internal use only.
write_conflicts	**bigint**	Internal use only.
unique_constraint_violations	**bigint**	Total number of unique constraint violations.

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Memory-Optimized Table Dynamic Management Views (Transact-SQL)

Object Related Dynamic Management Views and Functions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management objects in SQL Server.

sys.dm_db_incremental_stats_properties	sys.dm_db_stats_histogram
sys.dm_db_stats_properties	sys.dm_sql_referenced_entities
sys.dm_sql_referencing_entities	

See Also

Dynamic Management Views and Functions (Transact-SQL)
System Views (Transact-SQL)

sys.dm_db_incremental_stats_properties (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns properties of incremental statistics for the specified database object (table) in the current SQL Server database. The use of `sys.dm_db_incremental_stats_properties` (which contains a partition number) is similar to `sys.dm_db_stats_properties` which is used for non-incremental statistics.

This function was introduced in SQL Server 2014 (12.x) Service Pack 2 and SQL Server 2016 (13.x) Service Pack 1.

Syntax

```
sys.dm_db_incremental_stats_properties (object_id, stats_id)
```

Arguments

object_id
Is the ID of the object in the current database for which properties of one of its incremental statistics is requested. *object_id* is **int**.

stats_id
Is the ID of statistics for the specified *object_id*. The statistics ID can be obtained from the sys.stats dynamic management view. *stats_id* is **int**.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
object_id	**int**	ID of the object (table) for which to return the properties of the statistics object.
stats_id	**int**	ID of the statistics object. Is unique within the table. For more information, see sys.stats (Transact-SQL).
partition_number	**int**	Number of the partition containing the portion of the table.
last_updated	**datetime2**	Date and time the statistics object was last updated. For more information, see the Remarks section in this page.

COLUMN NAME	DATA TYPE	DESCRIPTION
rows	**bigint**	Total number of rows in the table when statistics were last updated. If the statistics are filtered or correspond to a filtered index, the number of rows might be less than the number of rows in the table.
rows_sampled	**bigint**	Total number of rows sampled for statistics calculations.
steps	**int**	Number of steps in the histogram. For more information, see DBCC SHOW_STATISTICS (Transact-SQL).
unfiltered_rows	**bigint**	Total number of rows in the table before applying the filter expression (for filtered statistics). If statistics are not filtered, unfiltered_rows is equal to the value returns in the rows column.
modification_counter	**bigint**	Total number of modifications for the leading statistics column (the column on which the histogram is built) since the last time statistics were updated. This column does not contain information for memory-optimized tables.

Remarks

`sys.dm_db_incremental_stats_properties` returns an empty rowset under any of the following conditions:

- `object_id` or `stats_id` is NULL.
- The specified object is not found or does not correspond to a table with incremental statistics.
- The specified statistics ID does not correspond to existing statistics for the specified object ID.
- The current user does not have permissions to view the statistics object.

  This behavior allows for the safe usage of `sys.dm_db_incremental_stats_properties` when cross applied to rows in views such as `sys.objects` and `sys.stats`. This method can return properties for the statistics that correspond to each partition. To see the properties for the merged statistics combined across all partitions, use the sys.dm_db_stats_properties instead.

Statistics update date is stored in the statistics blob object together with the histogram and density vector, not in the metadata. When no data is read to generate statistics data, the statistics blob is not created, the date is not available, and the *last_updated* column is NULL. This is the case for filtered statistics for which the predicate does not return any rows, or for new empty tables.

Permissions

Requires that the user has select permissions on statistics columns or the user owns the table or the user is a member of the `sysadmin` fixed server role, the `db_owner` fixed database role, or the `db_ddladmin` fixed database role.

Examples

A. Simple example

The following example returns the statistics for the `PartitionTable` table described in the topic Create Partitioned Tables and Indexes.

```
SELECT * FROM sys.dm_db_incremental_stats_properties (object_id('PartitionTable'), 1);
```

For additional usage suggestions, see sys.dm_db_stats_properties.

See Also

DBCC SHOW_STATISTICS (Transact-SQL)
sys.stats (Transact-SQL)
Object Related Dynamic Management Views and Functions (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)
sys.dm_db_stats_properties
sys.dm_db_stats_histogram (Transact-SQL)

sys.dm_db_stats_histogram (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the statistics histogram for the specified database object (table or indexed view) in the current SQL Server database. Similar to `DBCC SHOW_STATISTICS WITH HISTOGRAM`.

> **NOTE**
>
> This DMF is available starting with SQL Server 2016 (13.x) SP1 CU2

Syntax

```
sys.dm_db_stats_histogram (object_id, stats_id)
```

Arguments

object_id
Is the ID of the object in the current database for which properties of one of its statistics is requested. *object_id* is **int**.

stats_id
Is the ID of statistics for the specified *object_id*. The statistics ID can be obtained from the sys.stats dynamic management view. *stats_id* is **int**.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
object_id	**int**	ID of the object (table or indexed view) for which to return the properties of the statistics object.
stats_id	**int**	ID of the statistics object. Is unique within the table or indexed view. For more information, see sys.stats (Transact-SQL).
step_number	**int**	The number of step in the histogram.
range_high_key	**sql_variant**	Upper bound column value for a histogram step. The column value is also called a key value.
range_rows	**real**	Estimated number of rows whose column value falls within a histogram step, excluding the upper bound.

COLUMN NAME	DATA TYPE	DESCRIPTION
equal_rows	**real**	Estimated number of rows whose column value equals the upper bound of the histogram step.
distinct_range_rows	**bigint**	Estimated number of rows with a distinct column value within a histogram step, excluding the upper bound.
average_range_rows	**real**	Average number of rows with duplicate column values within a histogram step, excluding the upper bound (`RANGE_ROWS / DISTINCT_RANGE_ROWS` for `DISTINCT_RANGE_ROWS > 0`).

Remarks

The resultset for `sys.dm_db_stats_histogram` returns information similar to `DBCC SHOW_STATISTICS WITH HISTOGRAM` and also includes `object_id`, `stats_id`, and `step_number`.

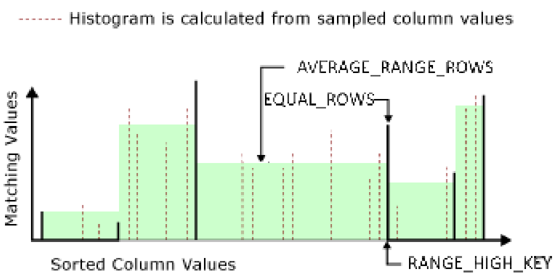
Because the column `range_high_key` is a sql_variant data type, you may need to use `CAST` or `CONVERT` if a predicate does comparison with a non-string constant.

Histogram

A histogram measures the frequency of occurrence for each distinct value in a data set. The query optimizer computes a histogram on the column values in the first key column of the statistics object, selecting the column values by statistically sampling the rows or by performing a full scan of all rows in the table or view. If the histogram is created from a sampled set of rows, the stored totals for number of rows and number of distinct values are estimates and do not need to be whole integers.

To create the histogram, the query optimizer sorts the column values, computes the number of values that match each distinct column value and then aggregates the column values into a maximum of 200 contiguous histogram steps. Each step includes a range of column values followed by an upper bound column value. The range includes all possible column values between boundary values, excluding the boundary values themselves. The lowest of the sorted column values is the upper boundary value for the first histogram step.

The following diagram shows a histogram with six steps. The area to the left of the first upper boundary value is the first step.



For each histogram step:

- Bold line represents the upper boundary value (*range_high_key*) and the number of times it occurs (*equal_rows*)
- Solid area left of *range_high_key* represents the range of column values and the average number of times each column value occurs (*average_range_rows*). The *average_range_rows* for the first histogram step is

always 0.

- Dotted lines represent the sampled values used to estimate total number of distinct values in the range (*distinct_range_rows*) and total number of values in the range (*range_rows*). The query optimizer uses *range_rows* and *distinct_range_rows* to compute *average_range_rows* and does not store the sampled values.

  The query optimizer defines the histogram steps according to their statistical significance. It uses a maximum difference algorithm to minimize the number of steps in the histogram while maximizing the difference between the boundary values. The maximum number of steps is 200. The number of histogram steps can be fewer than the number of distinct values, even for columns with fewer than 200 boundary points. For example, a column with 100 distinct values can have a histogram with fewer than 100 boundary points.

Permissions

Requires that the user has select permissions on statistics columns or the user owns the table or the user is a member of the `sysadmin` fixed server role, the `db_owner` fixed database role, or the `db_ddladmin` fixed database role.

Examples

A. Simple example

The following example creates and populates a simple table. Then creates statistics on the `Country_Name` column.

```
CREATE TABLE Country
(Country_ID int IDENTITY PRIMARY KEY,
Country_Name varchar(120) NOT NULL);
INSERT Country (Country_Name) VALUES ('Canada'), ('Denmark'), ('Iceland'), ('Peru');

CREATE STATISTICS Country_Stats
    ON Country (Country_Name) ;
```

The primary key occupies `stat_id` number 1, so call `sys.dm_db_stats_histogram` for `stat_id` number 2, to return the statistics histogram for the `Country` table.

```
SELECT * FROM sys.dm_db_stats_histogram(OBJECT_ID('Country'), 2);
```

B. Useful query:

```
SELECT hist.step_number, hist.range_high_key, hist.range_rows,
    hist.equal_rows, hist.distinct_range_rows, hist.average_range_rows
FROM sys.stats AS s
CROSS APPLY sys.dm_db_stats_histogram(s.[object_id], s.stats_id) AS hist
WHERE s.[name] = N'<statistic_name>';
```

C. Useful query:

The following example selects from table `Country` with a predicate on column `Country_Name`.

```
SELECT * FROM Country
WHERE Country_Name = 'Canada';
```

The following example looks at the previously created statistic on table `Country` and column `Country_Name` for the histogram step matching the predicate in the query above.

```
SELECT ss.name, ss.stats_id, shr.steps, shr.rows, shr.rows_sampled,
    shr.modification_counter, shr.last_updated, sh.range_rows, sh.equal_rows
FROM sys.stats ss
INNER JOIN sys.stats_columns sc
    ON ss.stats_id = sc.stats_id AND ss.object_id = sc.object_id
INNER JOIN sys.all_columns ac
    ON ac.column_id = sc.column_id AND ac.object_id = sc.object_id
CROSS APPLY sys.dm_db_stats_properties(ss.object_id, ss.stats_id) shr
CROSS APPLY sys.dm_db_stats_histogram(ss.object_id, ss.stats_id) sh
WHERE ss.[object_id] = OBJECT_ID('Country')
    AND ac.name = 'Country_Name'
    AND sh.range_high_key = CAST('Canada' AS CHAR(8));
```

See Also

DBCC SHOW_STATISTICS (Transact-SQL)
Object Related Dynamic Management Views and Functions (Transact-SQL)
sys.dm_db_stats_properties (Transact-SQL)

sys.dm_db_stats_properties (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns properties of statistics for the specified database object (table or indexed view) in the current SQL Server database. For partitioned tables, see the similar sys.dm_db_incremental_stats_properties.

Syntax

```
sys.dm_db_stats_properties (object_id, stats_id)
```

Arguments

object_id
Is the ID of the object in the current database for which properties of one of its statistics is requested. *object_id* is **int**.

stats_id
Is the ID of statistics for the specified *object_id*. The statistics ID can be obtained from the sys.stats dynamic management view. *stats_id* is **int**.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
object_id	**int**	ID of the object (table or indexed view) for which to return the properties of the statistics object.
stats_id	**int**	ID of the statistics object. Is unique within the table or indexed view. For more information, see sys.stats (Transact-SQL).
last_updated	**datetime2**	Date and time the statistics object was last updated. For more information, see the Remarks section in this page.
rows	**bigint**	Total number of rows in the table or indexed view when statistics were last updated. If the statistics are filtered or correspond to a filtered index, the number of rows might be less than the number of rows in the table.
rows_sampled	**bigint**	Total number of rows sampled for statistics calculations.

COLUMN NAME	DATA TYPE	DESCRIPTION
steps	**int**	Number of steps in the histogram. For more information, see DBCC SHOW_STATISTICS (Transact-SQL).
unfiltered_rows	**bigint**	Total number of rows in the table before applying the filter expression (for filtered statistics). If statistics are not filtered, unfiltered_rows is equal to the value returns in the rows column.
modification_counter	**bigint**	Total number of modifications for the leading statistics column (the column on which the histogram is built) since the last time statistics were updated. Memory-optimized tables: starting SQL Server 2016 (13.x) and in Azure SQL Database this column contains: total number of modifications for the table since the last time statistics were updated or the database was restarted.
persisted_sample_percent	**float**	Persisted sample percentage used for statistic updates that do not explicitly specify a sampling percentage. If value is zero, then no persisted sample percentage is set for this statistic. **Applies to:** SQL Server 2016 (13.x) SP1 CU4

Remarks

sys.dm_db_stats_properties returns an empty rowset under any of the following conditions:

- **object_id** or **stats_id** is NULL.
- The specified object is not found or does not correspond to a table or indexed view.
- The specified statistics ID does not correspond to existing statistics for the specified object ID.
- The current user does not have permissions to view the statistics object.

  This behavior allows for the safe usage of **sys.dm_db_stats_properties** when cross applied to rows in views such as **sys.objects** and **sys.stats**.

Statistics update date is stored in the statistics blob object together with the histogram and density vector, not in the metadata. When no data is read to generate statistics data, the statistics blob is not created, the date is not available, and the *last_updated* column is NULL. This is the case for filtered statistics for which the predicate does not return any rows, or for new empty tables.

Permissions

Requires that the user has select permissions on statistics columns or the user owns the table or the user is a member of the `sysadmin` fixed server role, the `db_owner` fixed database role, or the `db_ddladmin` fixed database role.

Examples

A. Simple example

The following example returns the statistics for the `Person.Person` table in the AdventureWorks database.

```
SELECT * FROM sys.dm_db_stats_properties (object_id('Person.Person'), 1);
```

B. Returning all statistics properties for a table

The following example returns properties of all statistics that exist for the table TEST.

```
SELECT sp.stats_id, name, filter_definition, last_updated, rows, rows_sampled, steps, unfiltered_rows,
modification_counter
FROM sys.stats AS stat
CROSS APPLY sys.dm_db_stats_properties(stat.object_id, stat.stats_id) AS sp
WHERE stat.object_id = object_id('TEST');
```

C. Returning statistics properties for frequently modified objects

The following example returns all tables, indexed views, and statistics in the current database for which the leading column was modified more than 1000 times since the last statistics update.

```
SELECT obj.name, obj.object_id, stat.name, stat.stats_id, last_updated, modification_counter
FROM sys.objects AS obj
INNER JOIN sys.stats AS stat ON stat.object_id = obj.object_id
CROSS APPLY sys.dm_db_stats_properties(stat.object_id, stat.stats_id) AS sp
WHERE modification_counter > 1000;
```

See Also

DBCC SHOW_STATISTICS (Transact-SQL)
sys.stats (Transact-SQL)
Object Related Dynamic Management Views and Functions (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)
sys.dm_db_incremental_stats_properties (Transact-SQL)
sys.dm_db_stats_histogram (Transact-SQL)

sys.dm_sql_referenced_entities (Transact-SQL)

5/4/2018 • 10 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row for each user-defined entity referenced by name in the definition of the specified referencing entity in SQL Server. A dependency between two entities is created when one user-defined entity, called the *referenced entity*, appears by name in a persisted SQL expression of another user-defined entity, called the *referencing entity*. For example, if a stored procedure is the specified referencing entity, this function returns all user-defined entities that are referenced in the stored procedure such as tables, views, user-defined types (UDTs), or other stored procedures.

You can use this dynamic management function to report on the following types of entities referenced by the specified referencing entity:

- Schema-bound entities
- Non-schema-bound entities
- Cross-database and cross-server entities
- Column-level dependencies on schema-bound and non-schema-bound entities
- User-defined types (alias and CLR UDT)
- XML schema collections
- Partition functions

Applies to: SQL Server (SQL Server 2008 through SQL Server 2017), SQL Database.

Syntax

```
sys.dm_sql_referenced_entities (
    ' [ schema_name. ] referencing_entity_name ' , ' <referencing_class> ' )

<referencing_class> ::=
{
    OBJECT
  | DATABASE_DDL_TRIGGER
  | SERVER_DDL_TRIGGER
}
```

Arguments

[*schema_name*.] *referencing_entity_name*
Is the name of the referencing entity. *schema_name* is required when the referencing class is OBJECT.

schema_name.referencing_entity_name is **nvarchar(517)**.

<*referencing_class*> ::= { OBJECT | DATABASE_DDL_TRIGGER | SERVER_DDL_TRIGGER }
Is the class of the specified referencing entity. Only one class can be specified per statement.

<*referencing_class*> is **nvarchar(60)**.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
referencing_minor_id	**int**	Column ID when the referencing entity is a column; otherwise 0. Is not nullable.
referenced_server_name	**sysname**	Name of the server of the referenced entity. This column is populated for cross-server dependencies that are made by specifying a valid four-part name. For information about multipart names, see Transact-SQL Syntax Conventions (Transact-SQL). NULL for non-schema-bound dependencies for which the entity was referenced without specifying a four-part name. NULL for schema-bound entities because they must be in the same database and therefore can only be defined using a two-part (*schema.object*) name.
referenced_database_name	**sysname**	Name of the database of the referenced entity. This column is populated for cross-database or cross-server references that are made by specifying a valid three-part or four-part name. NULL for non-schema-bound references when specified using a one-part or two-part name. NULL for schema-bound entities because they must be in the same database and therefore can only be defined using a two-part (*schema.object*) name.
referenced_schema_name	**sysname**	Schema in which the referenced entity belongs. NULL for non-schema-bound references in which the entity was referenced without specifying the schema name. Never NULL for schema-bound references.
referenced_entity_name	**sysname**	Name of the referenced entity. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
referenced_minor_name	**sysname**	Column name when the referenced entity is a column; otherwise NULL. For example, referenced_minor_name is NULL in the row that lists the referenced entity itself. A referenced entity is a column when a column is identified by name in the referencing entity, or when the parent entity is used in a SELECT * statement.
referenced_id	**int**	ID of the referenced entity. When referenced_minor_id is not 0, referenced_id is the entity in which the column is defined. Always NULL for cross-server references. NULL for cross-database references when the ID cannot be determined because the database is offline or the entity cannot be bound. NULL for references within the database if the ID cannot be determined. For non-schema-bound references, the ID cannot be resolved when the referenced entity does not exist in the database or when the name resolution is caller dependent. In the latter case, is_caller_dependent is set to 1. Never NULL for schema-bound references.
referenced_minor_id	**int**	Column ID when the referenced entity is a column; otherwise, 0. For example, referenced_minor_is is 0 in the row that lists the referenced entity itself. For non-schema-bound references, column dependencies are reported only when all referenced entities can be bound. If any referenced entity cannot be bound, no column-level dependencies are reported and referenced_minor_id is 0. See Example D.
referenced_class	**tinyint**	Class of the referenced entity. 1 = Object or column 6 = Type 10 = XML schema collection 21 = Partition function

COLUMN NAME	DATA TYPE	DESCRIPTION
referenced_class_desc	**nvarchar(60)**	Description of class of referenced entity. OBJECT_OR_COLUMN TYPE XML_SCHEMA_COLLECTION PARTITION_FUNCTION
is_caller_dependent	**bit**	Indicates schema binding for the referenced entity occurs at run time; therefore, resolution of the entity ID depends on the schema of the caller. This occurs when the referenced entity is a stored procedure, extended stored procedure, or user-defined function called within an EXECUTE statement. 1 = The referenced entity is caller dependent and is resolved at run time. In this case, referenced_id is NULL. 0 = The referenced entity ID is not caller dependent. Always 0 for schema-bound references and for cross-database and cross-server references that explicitly specify a schema name. For example, a reference to an entity in the format `EXEC MyDatabase.MySchema.MyProc` is not caller dependent. However, a reference in the format `EXEC MyDatabase..MyProc` is caller dependent.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_ambiguous	**bit**	Indicates the reference is ambiguous and can resolve at run time to a user-defined function, a user-defined type (UDT), or an xquery reference to a column of type **xml**. For example, assume the statement `SELECT Sales.GetOrder() FROM Sales.MySales` is defined in a stored procedure. Until the stored procedure is executed, it is not known whether `Sales.GetOrder()` is a user-defined function in the `Sales` schema or column named `Sales` of type UDT with a method named `GetOrder()`. 1 = Reference to a user-defined function or column user-defined type (UDT) method is ambiguous. 0 = Reference is unambiguous or the entity can be successfully bound when the function is called. Always 0 for schema-bound references.
is_selected	**bit**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. 1 = The object or column is selected.
is_updated	**bit**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. 1 = The object or column is modified.
is_select_all	**bit**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. 1 = The object is used in a SELECT * clause (object-level only).
is_all_columns_found	**bit**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. 1 = All column dependencies for the object could be found. 0 = Column dependencies for the object could not be found.
is_insert_all	**bit**	**Applies to**: SQL Server 2016 (13.x) through SQL Server 2017. 1 = The object is used in an INSERT statement without a column list (object-level only).

COLUMN NAME	DATA TYPE	DESCRIPTION
is_incomplete	**bit**	**Applies to**: SQL Server 2016 (13.x) SP2 through SQL Server 2017. 1 = The object or column has a binding error and is incomplete.

Exceptions

Returns an empty result set under any of the following conditions:

- A system object is specified.
- The specified entity does not exist in the current database.
- The specified entity does not reference any entities.
- An invalid parameter is passed.

  Returns an error when the specified referencing entity is a numbered stored procedure.

  Returns error 2020 when column dependencies cannot be resolved. This error does not prevent the query from returning object-level dependencies.

Remarks

This function can be executed in the context of the any database to return the entities that reference a server-level DDL trigger.

The following table lists the types of entities for which dependency information is created and maintained. Dependency information is not created or maintained for rules, defaults, temporary tables, temporary stored procedures, or system objects.

ENTITY TYPE	REFERENCING ENTITY	REFERENCED ENTITY
Table	Yes*	Yes
View	Yes	Yes
Transact-SQL stored procedure**	Yes	Yes
CLR stored procedure	No	Yes
Transact-SQL user-defined function	Yes	Yes
CLR user-defined function	No	Yes
CLR trigger (DML and DDL)	No	No
Transact-SQL DML trigger	Yes	No
Transact-SQL database-level DDL trigger	Yes	No
Transact-SQL server-level DDL trigger	Yes	No

ENTITY TYPE	REFERENCING ENTITY	REFERENCED ENTITY
Extended stored procedures	No	Yes
Queue	No	Yes
Synonym	No	Yes
Type (alias and CLR user-defined type)	No	Yes
XML schema collection	No	Yes
Partition function	No	Yes

* A table is tracked as a referencing entity only when it references a Transact-SQL module, user-defined type, or XML schema collection in the definition of a computed column, CHECK constraint, or DEFAULT constraint.

** Numbered stored procedures with an integer value greater than 1 are not tracked as either a referencing or referenced entity.

Permissions

Requires SELECT permission on sys.dm_sql_referenced_entities and VIEW DEFINITION permission on the referencing entity. By default, SELECT permission is granted to public. Requires VIEW DEFINITION permission on the database or ALTER DATABASE DDL TRIGGER permission on the database when the referencing entity is a database-level DDL trigger. Requires VIEW ANY DEFINITION permission on the server when the referencing entity is a server-level DDL trigger.

Examples

A. Returning entities that are referenced by a database-level DDL trigger

The following example returns the entities (tables and columns) that are referenced by the database-level DDL trigger `ddlDatabaseTriggerLog`.

```
USE AdventureWorks2012;
GO
SELECT referenced_schema_name, referenced_entity_name, referenced_minor_name,
    referenced_minor_id, referenced_class_desc
FROM sys.dm_sql_referenced_entities ('ddlDatabaseTriggerLog', 'DATABASE_DDL_TRIGGER');
GO
```

B. Returning entities that are referenced by an object

The following example returns the entities that are referenced by the user-defined function `dbo.ufnGetContactInformation`.

```
USE AdventureWorks2012;
GO
SELECT referenced_schema_name, referenced_entity_name, referenced_minor_name,
    referenced_minor_id, referenced_class_desc, is_caller_dependent, is_ambiguous
FROM sys.dm_sql_referenced_entities ('dbo.ufnGetContactInformation', 'OBJECT');
GO
```

C. Returning column dependencies

The following example creates the table `Table1` with the computed column `c` defined as the sum of columns `a` and `b`. The `sys.dm_sql_referenced_entities` view is then called. The view returns two rows, one for each column defined in the computed column.

```
USE AdventureWorks2012;
GO
CREATE TABLE dbo.Table1 (a int, b int, c AS a + b);
GO
SELECT referenced_schema_name AS schema_name,
    referenced_entity_name AS table_name,
    referenced_minor_name AS referenced_column,
    COALESCE(COL_NAME(OBJECT_ID(N'dbo.Table1'),referencing_minor_id), 'N/A') AS referencing_column_name
FROM sys.dm_sql_referenced_entities ('dbo.Table1', 'OBJECT');
GO

-- Remove the table.
DROP TABLE dbo.Table1;
GO
```

Here is the result set.

```
schema_name table_name referenced_column referencing_column
----------- ---------- ----------------- ------------------
dbo         Table1     a                 c
dbo         Table1     b                 c
```

D. Returning non-schema-bound column dependencies

The following example drops `Table1` and creates `Table2` and stored procedure `Proc1`. The procedure references `Table2` and the nonexistent table `Table1`. The view `sys.dm_sql_referenced_entities` is run with the stored procedure specified as the referencing entity. The result set shows one row for `Table1` and 3 rows for `Table2`. Because `Table1` does not exist, the column dependencies cannot be resolved and error 2020 is returned. The `is_all_columns_found` column returns 0 for `Table1` indicating that there were columns that could not be discovered.

```
USE AdventureWorks2012;
GO
IF OBJECT_ID ( 'dbo.Table1', 'U' ) IS NOT NULL
    DROP TABLE dbo.Table1;
GO
CREATE TABLE dbo.Table2 (c1 int, c2 int);
GO
CREATE PROCEDURE dbo.Proc1 AS
    SELECT a, b, c FROM Table1;
    SELECT c1, c2 FROM Table2;
GO
SELECT referenced_id, referenced_entity_name AS table_name, referenced_minor_name AS referenced_column_name,
is_all_columns_found
FROM sys.dm_sql_referenced_entities ('dbo.Proc1', 'OBJECT');
GO
```

Here is the result set.

```
referenced_id table_name   referenced_column_name  is_all_columns_found
------------- ------------ ---------------------- --------------------
935674381     Table2       NULL                    1
935674381     Table2       C1                      1
935674381     Table2       C2                      1
NULL          Table1       NULL                    0

Msg 2020, Level 16, State 1, Line 1The dependencies reported for entity "dbo.Proc1" might not include
references to all columns. This is either because the entity references an object that does not exist or
because of an error in one or more statements in the entity.  Before rerunning the query, ensure that there
are no errors in the entity and that all objects referenced by the entity exist.
```

E. Demonstrating dynamic dependency maintenance

The following example extends Example D to show that dependencies are maintained dynamically. The example first re-creates `Table1`, which was dropped in Example D. Then `sys.dm_sql_referenced_entities` is run again with the stored procedure specified as the referencing entity. The result set shows that both tables and their respective columns defined in the stored procedure are returned. In addition, the `is_all_columns_found` column returns a 1 for all objects and columns.

```
USE AdventureWorks2012;
GO
CREATE TABLE Table1 (a int, b int, c AS a + b);
GO
SELECT referenced_id, referenced_entity_name AS table_name, referenced_minor_name as column_name,
is_all_columns_found
FROM sys.dm_sql_referenced_entities ('dbo.Proc1', 'OBJECT');
GO
DROP TABLE Table1, Table2;
DROP PROC Proc1;
GO
```

Here is the result set.

```
referenced_id table_name   referenced_column_name  is_all_columns_found
------------- ------------ ---------------------- --------------------
935674381     Table2       NULL                    1
935674381     Table2       c1                      1
935674381     Table2       c2                      1
967674495     Table1       NULL                    1
967674495     Table1       a                       1
967674495     Table1       b                       1
967674495     Table1       c                       1
```

F. Returning object or column usage

The following example returns the objects and column dependencies of the stored procedure `HumanResources.uspUpdateEmployeePersonalInfo`. This procedure updates the columns `NationalIDNumber`, `` BirthDate,``MaritalStatus ``, and `Gender` of the `Employee` table based on a specified `BusinessEntityID` value. Another stored procedure, `upsLogError` is defined in a TRY...CATCH block to capture any execution errors. The `is_selected`, `is_updated`, and `is_select_all` columns return information about how these objects and columns are used within the referencing object. The table and columns that are modified are indicated by a 1 in the is_updated column. The `BusinessEntityID` column is only selected and the stored procedure `uspLogError` is neither selected nor modified.

Applies to: SQL Server 2012 (11.x) through SQL Server 2017.

```
SELECT referenced_entity_name AS table_name, referenced_minor_name as column_name, is_selected, is_updated,
is_select_all
FROM sys.dm_sql_referenced_entities ('HumanResources.uspUpdateEmployeePersonalInfo', 'OBJECT');
```

Here is the result set.

```
table_name    column_name         is_selected is_updated is_select_all
------------- ------------------- ----------- ---------- -------------
uspLogError   NULL                0           0          0
Employee      NULL                0           1          0
Employee      BusinessEntityID    1           0          0
Employee      NationalIDNumber    0           1          0
Employee      BirthDate           0           1          0
Employee      MaritalStatus       0           1          0
Employee      Gender              0           1          0
```

See Also

sys.dm_sql_referencing_entities (Transact-SQL)
sys.sql_expression_dependencies (Transact-SQL)

sys.dm_sql_referencing_entities (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row for each entity in the current database that references another user-defined entity by name. A dependency between two entities is created when one entity, called the *referenced entity*, appears by name in a persisted SQL expression of another entity, called the *referencing entity*. For example, if a user-defined type (UDT) is specified as the referenced entity, this function returns each user-defined entity that reference that type by name in its definition. The function does not return entities in other databases that may reference the specified entity. This function must be executed in the context of the master database to return a server-level DDL trigger as a referencing entity.

You can use this dynamic management function to report on the following types of entities in the current database that reference the specified entity:

- Schema-bound or non-schema-bound entities
- Database-level DDL triggers
- Server-level DDL triggers

Applies to: SQL Server (SQL Server 2008 through SQL Server 2017), SQL Database.

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_sql_referencing_entities (
    ' schema_name.referenced_entity_name ' , ' <referenced_class> ' )

<referenced_class> ::=
{
    OBJECT
  | TYPE
  | XML_SCHEMA_COLLECTION
  | PARTITION_FUNCTION
}
```

Arguments

schema_name.referenced_entity_name
Is the name of the referenced entity.

schema_name is required except when the referenced class is PARTITION_FUNCTION.

schema_name.referenced_entity_name is **nvarchar(517)**.

<referenced_class> ::= { OBJECT | TYPE | XML_SCHEMA_COLLECTION | PARTITION_FUNCTION }
Is the class of the referenced entity. Only one class can be specified per statement.

<referenced_class> is **nvarchar**(60).

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
referencing_schema_name	**sysname**	Schema in which the referencing entity belongs. Is nullable. NULL for database-level and server-level DDL triggers.
referencing_entity_name	**sysname**	Name of the referencing entity. Is not nullable.
referencing_id	**int**	ID of the referencing entity. Is not nullable.
referencing_class	**tinyint**	Class of the referencing entity. Is not nullable. 1 = Object 12 = Database-level DDL trigger 13 = Server-level DDL trigger
referencing_class_desc	**nvarchar(60)**	Description of class of referencing entity. OBJECT DATABASE_DDL_TRIGGER SERVER_DDL_TRIGGER
is_caller_dependent	**bit**	Indicates the resolution of the referenced entity ID occurs at run time because it depends on the schema of the caller. 1 = The referencing entity has the potential to reference the entity; however, resolution of the referenced entity ID is caller dependent and cannot be determined. This occurs only for non-schema-bound references to a stored procedure, extended stored procedure, or user-defined function called in an EXECUTE statement. 0 = Referenced entity is not caller dependent.

Exceptions

Returns an empty result set under any of the following conditions:

- A system object is specified.
- The specified entity does not exist in the current database.

- The specified entity does not reference any entities.
- An invalid parameter is passed.

  Returns an error when the specified referenced entity is a numbered stored procedure.

Remarks

The following table lists the types of entities for which dependency information is created and maintained. Dependency information is not created or maintained for rules, defaults, temporary tables, temporary stored procedures, or system objects.

ENTITY TYPE	REFERENCING ENTITY	REFERENCED ENTITY
Table	Yes*	Yes
View	Yes	Yes
Transact-SQL stored procedure**	Yes	Yes
CLR stored procedure	No	Yes
Transact-SQL user-defined function	Yes	Yes
CLR user-defined function	No	Yes
CLR trigger (DML and DDL)	No	No
Transact-SQL DML trigger	Yes	No
Transact-SQL database-level DDL trigger	Yes	No
Transact-SQL server-level DDL trigger	Yes	No
Extended stored procedures	No	Yes
Queue	No	Yes
Synonym	No	Yes
Type (alias and CLR user-defined type)	No	Yes
XML schema collection	No	Yes
Partition function	No	Yes

* A table is tracked as a referencing entity only when it references a Transact-SQL module, user-defined type, or XML schema collection in the definition of a computed column, CHECK constraint, or DEFAULT constraint.

** Numbered stored procedures with an integer value greater than 1 are not tracked as either a referencing or referenced entity.

Permissions

SQL Server 2008 – SQL Server 2012 (11.x)

- Requires CONTROL permission on the referenced object. When the referenced entity is a partition function, CONTROL permission on the database is required.
- Requires SELECT permission on sys.dm_sql_referencing_entities. By default, SELECT permission is granted to public.

SQL Server 2014 (12.x) - SQL Server 2017

- Requires no permissions on the referenced object. Partial results can be returned if the user has VIEW DEFINITION on only some of the referencing entities.
- Requires VIEW DEFINITION on the object when the referencing entity is an object.
- Requires VIEW DEFINITION on the database when the referencing entity is a database-level DDL trigger.
- Requires VIEW ANY DEFINITION on the server when the referencing entity is a server-level DDL trigger.

Examples

A. Returning the entities that refer to a given entity

The following example returns the entities in the current database that refer to the specified table.

```
USE AdventureWorks2012;
GO
SELECT referencing_schema_name, referencing_entity_name, referencing_id, referencing_class_desc,
is_caller_dependent
FROM sys.dm_sql_referencing_entities ('Production.Product', 'OBJECT');
GO
```

B. Returning the entities that refer to a given type

The following example returns the entities that reference the alias type `dbo.Flag`. The result set shows that two stored procedures use this type. The `dbo.Flag` type is also used in the definition of several columns in the `HumanResources.Employee` table; however, because the type is not in the definition of a computed column, CHECK constraint, or DEFAULT constraint in the table, no rows are returned for the `HumanResources.Employee` table.

```
USE AdventureWorks2012;
GO
SELECT referencing_schema_name, referencing_entity_name, referencing_id, referencing_class_desc,
is_caller_dependent
FROM sys.dm_sql_referencing_entities ('dbo.Flag', 'TYPE');
GO
```

Here is the result set.

```
referencing_schema_name referencing_entity_name   referencing_id referencing_class_desc is_caller_dependent
----------------------- ------------------------  ------------- --------------------- -------------------
HumanResources          uspUpdateEmployeeHireInfo  1803153469    OBJECT_OR_COLUMN      0
HumanResources          uspUpdateEmployeeLogin     1819153526    OBJECT_OR_COLUMN      0
(2 row(s) affected)`
```

See Also

sys.dm_sql_referenced_entities (Transact-SQL)
sys.sql_expression_dependencies (Transact-SQL)

Query Notifications - sys.dm_qn_subscriptions

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the active query notifications subscriptions in the server. You can use this view to check for active subscriptions in the server or a specified database, or to check for a specified server principal.

COLUMN NAME	DATA TYPE	DESCRIPTION
id	**int**	ID of a subscription.
database_id	**int**	ID of the database in which the notification query was executed. This database stores information related to this subscription.
sid	**varbinary(85)**	Security ID of the server principal that created and owns this subscription.
object_id	**int**	ID of the internal table that stores information about subscription parameters.
created	**datetime**	Date and time that the subscription was created.
timeout	**int**	Time-out for the subscription in seconds. The notification will be flagged to fire after this time has elapsed. Note: The actual firing time may be greater than the specified time-out. However, if a change that invalidates the subscription occurs after the specified time-out, but before the subscription is fired, SQL Server ensures that firing occurs at the time that the change was made.
status	**int**	Indicates the status of the subscription. See the table under remarks for the list of codes.

Relationship Cardinalities

FROM	TO	ON	TYPE
sys.dm_qn_subscriptions	**sys.databases**	**database_id**	Many-to-one
sys.dm_qn_subscriptions	**sys.internal_tables**	**object_id**	Many-to-one

Remarks

The status code of 0 indicates an undefined status.

The following status codes indicate that a subscription fired because of a change:

CODE	MINOR STATUS	INFO
65798	Subscription fired because data changed	Subscription triggered by insert
65799	Subscription fired because data changed	Delete
65800	Subscription fired because data changed	Update
65801	Subscription fired because data changed	Merge
65802	Subscription fired because data changed	Truncate table
66048	Subscription fired because timeout expired	Undefined info mode
66315	Subscription fired because object changed	object or user was dropped
66316	Subscription fired because object changed	object was altered
66565	Subscription fired because database was detached or dropped	server or db restarted
66571	Subscription fired because database was detached or dropped	object or user was dropped
66572	Subscription fired because database was detached or dropped	object was altered
67341	subscription was triggered because of lack od resources on the server	subscription was triggered because of lack od resources on the server

The following status codes indicate that a subscription failed to be created:

CODE	MINOR STATUS	INFO
132609	Subscription creation failed because the statement is not supported	Query is too complex
132610	Subscription creation failed because the statement is not supported	Invalid statement for subscription
132611	Subscription creation failed because the statement is not supported	Invalid set options for subscription
132612	Subscription creation failed because the statement is not supported	Invalid isolation level

CODE	MINOR STATUS	INFO
132622	Subscription creation failed because the statement is not supported	used internally
132623	Subscription creation failed because the statement is not supported	over the template limit per table

The following status codes are used internally and are classed as check kill and init modes:

CODE	MINOR STATUS	INFO
198656	Used internally: check kill and init modes	Undefined info mode
198928	Subscription was destroyed	Subscription fired because db was attached
198929	Subscription was destroyed	Subscription fired because user was dropped
198930	Subscription was destroyed	Subscription was dropped because of a resubscription
198931	Subscription was destroyed	subscription was killed
199168	Subscription is active	Undefined info mode
199424	Subscription initialized but not yet active	Undefined info mode

Permissions

Requires VIEW SERVER STATE permission on server.

> **NOTE**
>
> If the user does not have VIEW SERVER STATE permission, this view returns information about subscriptions owned by current user.

Examples

A. Return active query notification subscriptions for the current user

The following example returns the active query notification subscriptions of the current user. If the user has VIEW SERVER STATE permissions, all active subscriptions in the server are returned.

```
SELECT id, database_id, sid, object_id, created, timeout, status
FROM sys.dm_qn_subscriptions;
GO
```

B. Returning active query notification subscriptions for a specified user

The following example returns the active query notification subscriptions subscribed by login `Ruth0`.

```
SELECT id, database_id, sid, object_id, created, timeout, status
FROM sys.dm_qn_subscriptions
WHERE sid = SUSER_SID('Ruth0');
GO
```

C. Returning internal table metadata for query notification subscriptions

The following example returns the internal table metadata for query notification subscriptions.

```
SELECT qn.id AS query_subscription_id
    ,it.name AS internal_table_name
    ,it.object_id AS internal_table_id
FROM sys.internal_tables AS it
JOIN sys.dm_qn_subscriptions AS qn ON it.object_id = qn.object_id
WHERE it.internal_type_desc = 'QUERY_NOTIFICATION';
GO
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Query Notifications Related Dynamic Management Views (Transact-SQL)
KILL QUERY NOTIFICATION SUBSCRIPTION (Transact-SQL)

Replication Related Dynamic Management Views (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management objects used by replication.

sys.dm_repl_articles	sys.dm_repl_schemas
sys.dm_repl_tranhash	sys.dm_repl_traninfo

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_repl_articles (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about database objects published as articles in a replication topology.

COLUMN NAME	DATA TYPE	DESCRIPTION
artcache_db_address	**varbinary(8)**	In-memory address of the cached database structure for the publication database.
artcache_table_address	**varbinary(8)**	In-memory address of the cached table structure for a published table article.
artcache_schema_address	**varbinary(8)**	In-memory address of the cached article schema structure for a published table article.
artcache_article_address	**varbinary(8)**	In-memory address of the cached article structure for a published table article.
artid	**bigint**	Uniquely identifies each entry within this table.
artfilter	**bigint**	ID of the stored procedure used to horizontally filter the article.
artobjid	**bigint**	ID of the published object.
artpubid	**bigint**	ID of the publication to which the article belongs.

COLUMN NAME	DATA TYPE	DESCRIPTION
artstatus	**tinyint**	Bitmask of the article options and status, which can be the bitwise logical OR result of one or more of these values: **1** = Article is active. **8** = Include the column name in INSERT statements. **16** = Use parameterized statements. **24** = Both include the column name in INSERT statements and use parameterized statements. For example, an active article using parameterized statements would have a value of 17 in this column. A value of 0 means that the article is inactive and no additional properties are defined.
arttype	**tinyint**	Type of article: **1** = Log-based article. **3** = Log-based article with manual filter. **5** = Log-based article with manual view. **7** = Log-based article with manual filter and manual view. **8** = Stored procedure execution. **24** = Serializable stored procedure execution. **32** = Stored procedure (schema only). **64** = View (schema only). **128** = Function (schema only).
wszArtdesttable	**nvarchar(514)**	Name of published object at the destination.
wszArtdesttableowner	**nvarchar(514)**	Owner of published object at the destination.
wszArtinscmd	**nvarchar(510)**	Command or stored procedure used for inserts.

COLUMN NAME	DATA TYPE	DESCRIPTION
cmdTypeIns	**int**	Call syntax for the insert stored procedure, and can be one of these values. **1** = CALL **2** = SQL **3** = NONE **7** = UNKNOWN
wszArtdelcmd	**nvarchar(510)**	Command or stored procedure used for deletes.
cmdTypeDel	**int**	Call syntax for the delete stored procedure, and can be one of these values. **0** = XCALL **1** = CALL **2** = SQL **3** = NONE **7** = UNKNOWN
wszArtupdcmd	**nvarchar(510)**	Command or stored procedure used for updates.
cmdTypeUpd	**int**	Call syntax for the update stored procedure, and can be one of these values. **0** = XCALL **1** = CALL **2** = SQL **3** = NONE **4** = MCALL **5** = VCALL **6** = SCALL **7** = UNKNOWN
wszArtpartialupdcmd	**nvarchar(510)**	Command or stored procedure used for partial updates.

COLUMN NAME	DATA TYPE	DESCRIPTION
cmdTypePartialUpd	**int**	Call syntax for the partial update stored procedure, and can be one of these values. **2** = SQL
numcol	**int**	Number of columns in the partition for a vertically filtered article.
artcmdtype	**tinyint**	Type of command currently being replicated, and can be one of these values. **1** = INSERT **2** = DELETE **3** = UPDATE **4** = UPDATETEXT **5** = none **6** = internal use only **7** = internal use only **8** = partial UPDATE
artgeninscmd	**nvarchar(510)**	INSERT command template based on the columns included in the article.
artgendelcmd	**nvarchar(510)**	DELETE command template, which can include the primary key or the columns included in the article, depending on the call syntax is used.
artgenupdcmd	**nvarchar(510)**	UPDATE command template, which can include the primary key, updated columns, or a complete column list depending on the call syntax is used.
artpartialupdcmd	**nvarchar(510)**	Partial UPDATE command template, which includes the primary key and updated columns.
artupdtxtcmd	**nvarchar(510)**	UPDATETEXT command template, which includes the primary key and updated columns.
artgenins2cmd	**nvarchar(510)**	INSERT command template used when reconciling an article during concurrent snapshot processing.
artgendel2cmd	**nvarchar(510)**	DELETE command template used when reconciling an article during concurrent snapshot processing.

COLUMN NAME	DATA TYPE	DESCRIPTION
fInReconcile	**tinyint**	Indicates whether an article is currently being reconciled during concurrent snapshot processing.
fPubAllowUpdate	**tinyint**	Indicates whether the publication allows updating subscription.
intPublicationOptions	**bigint**	Bitmap that specifies additional publishing options, where the bitwise option values are: **0x1** - Enabled for peer-to-peer replication. **0x2** - Publish only local changes. **0x4** - Enabled for non-SQL Server Subscribers.

Permissions

Requires VIEW DATABASE STATE permission on the publication database to call **dm_repl_articles**.

Remarks

Information is only returned for replicated database objects that are currently loaded in the replication article cache.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Replication Related Dynamic Management Views (Transact-SQL)

sys.dm_repl_schemas (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about table columns published by replication.

COLUMN NAME	DATA TYPE	DESCRIPTION
artcache_schema_address	**varbinary(8)**	In-memory address of the cached schema structure for the published table article.
tabid	**bigint**	ID of the replicated table.
indexid	**smallint**	ID of a clustered index on the published table.
idSch	**bigint**	ID of the table schema.
tabschema	**nvarchar(510)**	Name of the table schema.
ccTabschema	**smallint**	Character length of the table schema.
tabname	**nvarchar(510)**	Name of the published table.
ccTabname	**smallint**	Character length of the published table name.
rowsetid_delete	**bigint**	ID of the deleted row.
rowsetid_insert	**bigint**	ID of the inserted row.
num_pk_cols	**int**	Number of primary key columns.
pcitee	**binary(8000)**	Pointer to the query expression structure used to evaluate computed column.
re_numtextcols	**int**	Number of binary large object columns in the replicated table.
re_schema_lsn_begin	**binary(8000)**	Beginning log sequence number (LSN) of schema version logging.
re_schema_lsn_end	**binary(8000)**	Ending LSN of schema version logging.
re_numcols	**int**	Number of columns published.
re_colid	**int**	Column identifier at the Publisher.

COLUMN NAME	DATA TYPE	DESCRIPTION
re_awcName	**nvarchar(510)**	Name of the published column.
re_ccName	**smallint**	Number of characters in the column name.
re_pk	**tinyint**	Whether the published column is part of a primary key.
re_unique	**tinyint**	Whether the published column is part of a unique index.
re_maxlen	**smallint**	Maximum length of the published column.
re_prec	**tinyint**	Precision of the published column.
re_scale	**tinyint**	Scale of the published column.
re_collatid	**bigint**	Collation ID for published column.
re_xvtype	**smallint**	Type of the published column.
re_offset	**smallint**	Offset of the published column.
re_bitpos	**tinyint**	Bit position of the published column, in the byte vector.
re_fNullable	**tinyint**	Specifies whether the published column supports NULL values.
re_fAnsiTrim	**tinyint**	Specifies whether ANSI trim is used on the published column.
re_computed	**smallint**	Specifies whether the published column is a computed column.
se_rowsetid	**bigint**	ID of the rowset.
se_schema_lsn_begin	**binary(8000)**	Beginning LSN of schema version logging.
se_schema_lsn_end	**binary(8000)**	Ending LSN of schema version logging.
se_numcols	**int**	Number of columns.
se_colid	**int**	ID of the column at the Subscriber.
se_maxlen	**smallint**	Maximum length of the column.
se_prec	**tinyint**	Precision of the column.
se_scale	**tinyint**	Scale of the column.

COLUMN NAME	DATA TYPE	DESCRIPTION
se_collatid	**bigint**	Collation ID for column.
se_xvtype	**smallint**	Type of the column.
se_offset	**smallint**	Offset of the column.
se_bitpos	**tinyint**	Bit position of the column, in the byte vector.
se_fNullable	**tinyint**	Specifies whether the column supports NULL values.
se_fAnsiTrim	**tinyint**	Specifies whether ANSI trim is used on the column.
se_computed	**smallint**	Specifies whether the columnis a computed column.
se_nullBitInLeafRows	**int**	Specifies whether the column value is NULL.

Permissions

Requires VIEW DATABASE STATE permission on the publication database to call **dm_repl_schemas**.

Remarks

Information is only returned for replicated database objects that are currently loaded in the replication article cache.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Replication Related Dynamic Management Views (Transact-SQL)

sys.dm_repl_tranhash (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about transactions being replicated in a transactional publication.

COLUMN_NAME	DATA_TYPE	DESCRIPTION
buckets	**bigint**	Number of buckets in the hash table.
hashed_trans	**bigint**	Number of committed transactions replicated in the current batch.
completed_trans	**bigint**	Number of transactions competed so far.
compensated_trans	**bigint**	Number of transactions that contain partial rollbacks.
first_begin_lsn	**nvarchar(64)**	Earliest begin log sequence number (LSN) in the current batch.
last_commit_lsn	**nvarchar(64)**	Last commit LSN in the current batch.

Permissions

Requires VIEW DATABASE STATE permission on the publication database to call **dm_repl_tranhash**.

Remarks

Information is only returned for replicated database objects that are currently loaded in the replication article cache.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Replication Related Dynamic Management Views (Transact-SQL)

sys.dm_repl_traninfo (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information on each replicated or change data capture transaction.

COLUMN NAME	DATA TYPE	DESCRIPTION
fp2p_pub_exists	**tinyint**	If the transaction is in a database published using peer-to-peer transactional replication. If true, the value is 1; otherwise, it is 0.
db_ver	**int**	Database version.
comp_range_address	**varbinary(8)**	Defines a partial rollback range that must be skipped.
textinfo_address	**varbinary(8)**	In-memory address of the cached text information structure.
fsinfo_address	**varbinary(8)**	In-memory address of the cached filestream information structure.
begin_lsn	**nvarchar(64)**	Log sequence number (LSN) of the beginning log record for the transaction.
commit_lsn	**nvarchar(64)**	LSN of commit log record for the transaction.
dbid	**smallint**	Database ID.
rows	**int**	ID of the replicated command within the transaction.
xdesid	**nvarchar(64)**	Transaction ID.
artcache_table_address	**varbinary(8)**	In-memory address of the cached article table structure last used for this transaction.
server	**nvarchar(514)**	Server name.
server_len_in_bytes	**smallint**	Character length, in bytes, of the server name.
database	**nvarchar(514)**	Database name.

COLUMN NAME	DATA TYPE	DESCRIPTION
db_len_in_bytes	**smallint**	Character length, in bytes, of the database name.
originator	**nvarchar(514)**	Name of the server where the transaction originated.
originator_len_in_bytes	**smallint**	Character length, in bytes, of the server where the transaction originated.
orig_db	**nvarchar(514)**	Name of the database where the transaction originated.
orig_db_len_in_bytes	**smallint**	Character length, in bytes, of the database where the transaction originated.
cmds_in_tran	**int**	Number of replicated commands in the current transaction, which is used to determine when a logical transaction should be committed.
is_boundedupdate_singleton	**tinyint**	Specifies whether a unique column update affects only a single row.
begin_update_lsn	**nvarchar(64)**	LSN used in a unique column update.
delete_lsn	**nvarchar(64)**	LSN to delete as part of an update.
last_end_lsn	**nvarchar(64)**	Last LSN in a logical transaction.
fcomplete	**tinyint**	Specifies whether the command is a partial update.
fcompensated	**tinyint**	Specifies whether the transaction is involved in a partial rollback.
fprocessingtext	**tinyint**	Specifies whether the transaction includes a binary large data type column.
max_cmds_in_tran	**int**	Maximum number of commands in a logical transaction, as specified by the Log Reader Agent.
begin_time	**datetime**	Time the transaction began.
commit_time	**datetime**	Time the transaction was committed.
session_id	**int**	ID of the change data capture log scan session. This column maps to the **session_id** column in sys.dm_cdc_logscan_sessions.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_phase	**int**	Number that indicates the phase the session was in at the time the error occurred. This column maps to the **phase_number** column in sys.dm_cdc_errors.
is_known_cdc_tran	**bit**	Indicates the transaction is tracked by change data capture. 0 = Transaction replication transaction. 1 = Change data capture transaction.
error_count	**int**	Number of errors encountered.

Permissions

Requires VIEW DATABASE STATE permission on the publication database or on the database enabled for change data capture.

Remarks

Information is only returned for replicated database objects or tables enabled for change data capture that are currently loaded in the article cache.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Replication Related Dynamic Management Views (Transact-SQL)
Change Data Capture Related Dynamic Management Views (Transact-SQL)

Resource Governor Related Dynamic Management Views (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management views for the Resource Governor.

sys.dm_resource_governor_configuration (Transact-SQL)	sys.dm_resource_governor_resource_pools (Transact-SQL)
sys.dm_resource_governor_external_resource_pool_affinity (Transact-SQL)	sys.dm_resource_governor_resource_pool_volumes (Transact-SQL)
sys.dm_resource_governor_resource_pool_affinity (Transact-SQL)	sys.dm_resource_governor_workload_groups (Transact-SQL)

See Also

Resource Governor Catalog Views (Transact-SQL)
Resource Governor

sys.dm_resource_governor_configuration (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row that contains the current in-memory configuration state of Resource Governor.

COLUMN NAME	DATA TYPE	DESCRIPTION
classifier_function_id	**int**	The ID of the classifier function that is currently used by Resource Governor. Returns a value of 0 if no function is being used. Is not nullable. **Note:** This function is used to classify new requests and uses rules to route these requests to the appropriate workload group. For more information, see Resource Governor.
is_reconfiguration_pending	**bit**	Indicates whether or not changes to a group or pool were made with the ALTER RESOURCE GOVERNOR RECONFIGURE statement but have not been applied to the in-memory configuration. The value returned is one of: 0 - A reconfiguration statement is not required. 1 - A reconfiguration statement or server restart is required in order for pending configuration changes to be applied. **Note:** The value returned is always 0 when Resource Governor is disabled. Is not nullable.
max_outstanding_io_per_volume	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The maximum number of outstanding I/O per volume.

Remarks

This dynamic management view shows the in-memory configuration. To see the stored configuration metadata, use the corresponding catalog view.

The following example shows how to get and compare the stored metadata values and the in-memory values of the Resource Governor configuration.

```
USE master;
go
-- Get the stored metadata.
SELECT
object_schema_name(classifier_function_id) AS 'Classifier UDF schema in metadata',
object_name(classifier_function_id) AS 'Classifier UDF name in metadata'
FROM
sys.resource_governor_configuration;
go
-- Get the in-memory configuration.
SELECT
object_schema_name(classifier_function_id) AS 'Active classifier UDF schema',
object_name(classifier_function_id) AS 'Active classifier UDF name'
FROM
sys.dm_resource_governor_configuration;
go
```

Permissions

Requires VIEW SERVER STATE permission.

See Also

Dynamic Management Views and Functions (Transact-SQL)
sys.resource_governor_configuration (Transact-SQL)
Resource Governor

sys.dm_resource_governor_external_resource_pool_affinity (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Applies to: SQL Server 2016 (13.x) R Services (In-Database) and SQL Server 2017 (14.x) Machine Learning Services (In-Database)

Returns CPU affinity information about the current external resource pool configuration.

COLUMN NAME	DATA TYPE	DESCRIPTION
pool_id	**int**	The ID of the external resource pool. Is not nullable.
processor_group	**smallint**	The ID of the Windows logical processor group. Is not nullable.
cpu_mask	**bigint**	The binary mask representing the CPUs associated with this pool. Is not nullable.

Remarks

Pools that are created with an affinity of `AUTO` do not appear in this view because they have no affinity. For more information, see the CREATE EXTERNAL RESOURCE POOL (Transact-SQL) and ALTER EXTERNAL RESOURCE POOL (Transact-SQL) statements.

Permissions

Requires `VIEW SERVER STATE` permission.

See also

Resource governance for machine learning in SQL Server

sys.dm_resource_governor_resource_pool_affinity (Transact-SQL)

external scripts enabled Server Configuration Option

ALTER EXTERNAL RESOURCE POOL (Transact-SQL)

sys.dm_resource_governor_external_resource_pools (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the current external resource pool state, the current configuration of resource pools, and resource pool statistics.

Transact-SQL Syntax Conventions.

COLMN NAME	DATA TYPE	DESCRIPTION
external_pool_id	**int**	The ID of the resource pool. Is not nullable.
name	**sysname**	The name of the resource pool. Is not nullable.
pool_version	**int**	nternal version number.
max_cpu_percent	**int**	The current configuration for the maximum average CPU bandwidth allowed for all requests in the resource pool when there is CPU contention. Is not nullable.
max_processes	**int**	Maximum number of concurrent external processes. The default value, 0, specifies no limit. Is not nullable.
max_memory_percent	**int**	The current configuration for the percentage of total server memory that can be used by requests in this resource pool. Is not nullable.
statistics_start_time	**datetime**	The time when statistics was reset for this pool. Is not nullable.
peak_memory_kb	**bigint**	he maximum amount of memory used, in kilobytes, for the resource pool. Is not nullable.
write_io_count	**int**	The total write IOs issued since the Resource Govenor statistics were reset. Is not nullable.
read_io_count	**int**	The total read IOs issued since the Resource Govenor statistics were reset. Is not nullable.

COLMN NAME	DATA TYPE	DESCRIPTION
total_cpu_kernel_ms	**bigint**	The cumulative CPU user time in milliseconds since the Resource Govenor statistics were reset. Is not nullable.
total_cpu_user_ms	**bigint**	The cumulative CPU user time in milliseconds since the Resource Govenor statistics were reset. Is not nullable.
active_processes_count	**int**	The number of external processes running at the moment of the request. Is not nullable.

Permissions

Requires `VIEW SERVER STATE` permission.

See Also

sys.dm_resource_governor_external_resource_pool_affinity (Transact-SQL)

sys.dm_resource_governor_resource_pool_affinity (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Tracks resource pool affinity.

Transact-SQL Syntax Conventions.

COLMN NAME	DATA TYPE	DESCRIPTION
Pool_id	**int**	The ID of the resource pool. Is not nullable.
Processor_group	**smallint**	The ID of the Windows logical processor group. Is not nullable.
Scheduler_mask	**bigint**	The binary mask representing the schedulers associated with this pool. Is not nullable.

Remarks

Pools that are created with an affinity of AUTO will not appear in this view because they have no affinity. For more information, see the CREATE RESOURCE POOL (Transact-SQL) and ALTER RESOURCE POOL (Transact-SQL) statements.

See Also

sys.dm_resource_governor_external_resource_pool_affinity (Transact-SQL)

sys.dm_resource_governor_resource_pools (Transact-SQL)

5/4/2018 • 6 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the current resource pool state, the current configuration of resource pools, and resource pool statistics.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_resource_governor_resource_pools**.

COLUMN NAME	DATA TYPE	DESCRIPTION
pool_id	**int**	The ID of the resource pool. Is not nullable.
name	**sysname**	The name of the resource pool. Is not nullable.
statistics_start_time	**datetime**	The time when statistics was reset for this pool. Is not nullable.
total_cpu_usage_ms	**bigint**	The cumulative CPU usage in milliseconds since the Resource Govenor statistics were reset. Is not nullable.
cache_memory_kb	**bigint**	The current total cache memory usage in kilobytes. Is not nullable.
compile_memory_kb	**bigint**	The current total stolen memory usage in kilobytes (KB). The majority of this usage would be for compile and optimization, but it can also include other memory users. Is not nullable.
used_memgrant_kb	**bigint**	The current total used (stolen) memory from memory grants. Is not nullable.
total_memgrant_count	**bigint**	The cumulative count of memory grants in this resource pool. Is not nullable.
total_memgrant_timeout_count	**bigint**	The cumulative count of memory grant time-outs in this resource pool. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
active_memgrant_count	**int**	The current count of memory grants. Is not nullable.
active_memgrant_kb	**bigint**	The sum, in kilobytes (KB), of current memory grants. Is not nullable.
memgrant_waiter_count	**int**	The count of queries currently pending on memory grants. Is not nullable.
max_memory_kb	**bigint**	The maximum amount of memory, in kilobytes, that the resource pool can have. This is based on the current settings and server state. Is not nullable.
used_memory_kb	**bigint**	The amount of memory used, in kilobytes, for the resource pool. Is not nullable.
target_memory_kb	**bigint**	The target amount of memory, in kilobytes, the resource pool is trying to attain. This is based on the current settings and server state. Is not nullable.
out_of_memory_count	**bigint**	The number of failed memory allocations in the pool since the Resource Govenor statistics were reset. Is not nullable.
min_cpu_percent	**int**	The current configuration for the guaranteed average CPU bandwidth for all requests in the resource pool when there is CPU contention. Is not nullable.
max_cpu_percent	**int**	The current configuration for the maximum average CPU bandwidth allowed for all requests in the resource pool when there is CPU contention. Is not nullable.
min_memory_percent	**int**	The current configuration for the guaranteed amount of memory for all requests in the resource pool when there is memory contention. This is not shared with other resource pools. Is not nullable.
max_memory_percent	**int**	The current configuration for the percentage of total server memory that can be used by requests in this resource pool. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
cap_cpu_percent	**int**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Hard cap on the CPU bandwidth that all requests in the resource pool will receive. Limits the maximum CPU bandwidth level to the specified level. The allowed range for value is from 1 through 100. Is not nullable.
min_iops_per_volume	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The minimum IO per second (IOPS) per disk volume setting for this Pool. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0.
max_iops_per_volume	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The maximum IO per second (IOPS) per disk volume setting for this Pool. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0..
read_io_queued_total	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The total read IOs enqueued since the Resource Govenor was reset. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0.
read_io_issued_total	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The total read IOs issued since the Resource Govenor statistics were reset. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0.

COLUMN NAME	DATA TYPE	DESCRIPTION
read_io_completed_total	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The total read IOs completed since the Resource Govenor statistics were reset. Is not nullable.
read_io_throttled_total	**int**	The total read IOs throttled since the Resource Govenor statistics were reset. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0.
read_bytes_total	**bigint**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The total number of bytes read since the Resource Govenor statistics were reset. Is not nullable.
read_io_stall_total_ms	**bigint**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. Total time (in milliseconds) between read IO arrival and completion. Is not nullable.
read_io_stall_queued_ms	**bigint**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. Total time (in milliseconds) between read IO arrival and issue. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0. To determine if the IO setting for the pool is causing latency, subtract **read_io_stall_queued_ms** from **read_io_stall_total_ms**.
write_io_queued_total	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The total write IOs enqueued since the Resource Govenor statistics were reset. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0.

COLUMN NAME	DATA TYPE	DESCRIPTION
write_io_issued_total	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The total write IOs issued since the Resource Govenor statistics were reset. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0.
write_io_completed_total	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The total write IOs completed since the Resource Govenor statistics were reset. Is not nullable
write_io_throttled_total	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The total write IOs throttled since the Resource Govenor statistics were reset. Is not nullable
write_bytes_total	**bigint**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. The total number of bytes written since the Resource Govenor statistics were reset. Is not nullable.
write_io_stall_total_ms	**bigint**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. Total time (in milliseconds) between write IO arrival and completion. Is not nullable.
write_io_stall_queued_ms	**bigint**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. Total time (in milliseconds) between write IO arrival and issue. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0. This is the delay introduced by IO Resource Governance.

COLUMN NAME	DATA TYPE	DESCRIPTION
io_issue_violations_total	**int**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. Total IO issue violations. That is, the number of times when the rate of IO issue was lower than the reserved rate. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0.
io_issue_delay_total_ms	**bigint**	**Applies to**: SQL Server 2014 (12.x) through SQL Server 2017. Total time (in milliseconds) between the scheduled issue and actual issue of IO. Is nullable. Null if the resource pool is not governed for IO. That is, the Resource Pool MIN_IOPS_PER_VOLUME and MAX_IOPS_PER_VOLUME settings are 0.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Remarks

Resource Governor workload groups and Resource Governor resource pools have a many-to-one mapping. As a result, many of the resource pool statistics are derived from the workload group statistics.

This dynamic management view shows the in-memory configuration. To see the stored configuration metadata, use the sys.resource_governor_resource_pools catalog view.

Permissions

Requires VIEW SERVER STATE permission.

See Also

Dynamic Management Views and Functions (Transact-SQL)
sys.dm_resource_governor_workload_groups (Transact-SQL)
sys.resource_governor_resource_pools (Transact-SQL)
ALTER RESOURCE GOVERNOR (Transact-SQL)

sys.dm_resource_governor_resource_pool_volumes (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the current resource pool IO statistics for each disk volume. This information is also available at the resource pool level in sys.dm_resource_governor_resource_pools (Transact-SQL).

COLUMN NAME	DATA TYPE	DESCRIPTION
pool_id	**int**	The ID of the resource pool. Is not nullable.
volume_name	**sysname**	The name of the disk volue. Is not nullable.
read_io_queued_total	**int**	The total read IOs enqueued since the Resource Govenor is reset. Is not nullable.
read_io_issued_total	**int**	The total read IOs issued since the Resource Govenor statistics were reset. Is not nullable.
read_ios_completed_total	**int**	The total read IOs completed since the Resource Govenor statistics were reset. Is not nullable.
read_ios_throttled_total	**int**	The total read IOs throttled since the Resource Govenor statistics were reset. Is not nullable.
read_bytes_total	**bigint**	The total number of bytes read since the Resource Govenor statistics were reset. Is not nullable.
read_io_stall_total_ms	**bigint**	Total time (in milliseconds) between read IO arrival and completion. Is not nullable.
read_io_stall_queued_ms	**bigint**	Total time (in milliseconds) between read IO arrival and issue. This is the delay introduced by IO Resource Governance. Is not nullable.
write_io_queued_total	**int**	The total write IOs enqueued since the Resource Govenor statistics were reset. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
write_io_issued_total	**int**	The total write IOs issued since the Resource Govenor statistics were reset. Is not nullable.
write_io_completed_total	**int**	The total write IOs completed since the Resource Govenor statistics were reset. Is not nullable
write_io_throttled_total	**int**	The total write IOs throttled since the Resource Govenor statistics were reset. Is not nullable
write_bytes_total	**bigint**	The total number of bytes written since the Resource Govenor statistics were reset. Is not nullable.
write_io_stall_total_ms	**bigint**	Total time (in milliseconds) between write IO issue and completion. Is not nullable.
write_io_stall_queued_ms	**bigint**	Total time (in milliseconds) between write IO arrival and issue. This is the delay introduced by IO Resource Governance. Is not nullable.
io_issue_violations_total	**int**	Total IO issue violations. That is, the number of times when the rate of IO issue was lower than the reserved rate. Is not nullable.
io_issue_delay_total_ms	**bigint**	Total time (in milliseconds) between the scheduled issue and actual issue of IO. Is not nullable.

Permissions

Requires VIEW SERVER STATE permission.

See also

Dynamic Management Views and Functions (Transact-SQL)
sys.dm_resource_governor_workload_groups (Transact-SQL)
sys.resource_governor_resource_pools (Transact-SQL)
ALTER RESOURCE GOVERNOR (Transact-SQL)

sys.dm_resource_governor_workload_groups (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns workload group statistics and the current in-memory configuration of the workload group. This view can be joined with sys.dm_resource_governor_resource_pools to get the resource pool name.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_resource_governor_workload_groups**.

COLUMN NAME	DATA TYPE	DESCRIPTION
group_id	**int**	ID of the workload group. Is not nullable.
name	**sysname**	Name of the workload group. Is not nullable.
pool_id	**int**	ID of the resource pool. Is not nullable.
external_pool_id	**int**	**Applies to**: SQL Server 2016 (13.x) through SQL Server 2017. ID of the external resource pool. Is not nullable.
statistics_start_time	**datetime**	Time that statistics collection was reset for the workload group. Is not nullable.
total_request_count	**bigint**	Cumulative count of completed requests in the workload group. Is not nullable.
total_queued_request_count	**bigint**	Cumulative count of requests queued after the GROUP_MAX_REQUESTS limit was reached. Is not nullable.
active_request_count	**int**	Current request count. Is not nullable.
queued_request_count	**int**	Current queued request count. Is not nullable.
total_cpu_limit_violation_count	**bigint**	Cumulative count of requests exceeding the CPU limit. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
total_cpu_usage_ms	**bigint**	Cumulative CPU usage, in milliseconds, by this workload group. Is not nullable.
max_request_cpu_time_ms	**bigint**	Maximum CPU usage, in milliseconds, for a single request. Is not nullable. **Note:** This is a measured value, unlike request_max_cpu_time_sec, which is a configurable setting. For more information, see CPU Threshold Exceeded Event Class.
blocked_task_count	**int**	Current count of blocked tasks. Is not nullable.
total_lock_wait_count	**bigint**	Cumulative count of lock waits that occurred. Is not nullable.
total_lock_wait_time_ms	**bigint**	Cumulative sum of elapsed time, in milliseconds, a lock is held. Is not nullable.
total_query_optimization_count	**bigint**	Cumulative count of query optimizations in this workload group. Is not nullable.
total_suboptimal_plan_generation_count	**bigint**	Cumulative count of suboptimal plan generations that occurred in this workload group due to memory pressure. Is not nullable.
total_reduced_memgrant_count	**bigint**	Cumulative count of memory grants that reached the maximum query size limit. Is not nullable.
max_request_grant_memory_kb	**bigint**	Maximum memory grant size, in kilobytes, of a single request since the statistics were reset. Is not nullable.
active_parallel_thread_count	**bigint**	Current count of parallel thread usage. Is not nullable.
importance	**sysname**	Current configuration value for the relative importance of a request in this workload group. Importance is one of the following, with Medium being the default: Low, Medium, or High. Is not nullable.
request_max_memory_grant_percent	**int**	Current setting for the maximum memory grant, as a percentage, for a single request. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
request_max_cpu_time_sec	**int**	Current setting for maximum CPU use limit, in seconds, for a single request. Is not nullable.
request_memory_grant_timeout_sec	**int**	Current setting for memory grant time-out, in seconds, for a single request. Is not nullable.
group_max_requests	**int**	Current setting for the maximum number of concurrent requests. Is not nullable.
max_dop	**int**	Maximum degree of parallelism for the workload group. The default value, 0, uses global settings. Is not nullable.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Remarks

This dynamic management view shows the in-memory configuration. To see the stored configuration metadata, use the sys.resource_governor_workload_groups catalog view.

When ALTER RESOURCE GOVERNOR RESET STATISTICS is successfully executed, the following counters are reset: statistics_start_time, total_request_count, total_queued_request_count, total_cpu_limit_violation_count, total_cpu_usage_ms, max_request_cpu_time_ms, total_lock_wait_count, total_lock_wait_time_ms, total_query_optimization_count, total_suboptimal_plan_generation_count, total_reduced_memgrant_count, and max_request_grant_memory_kb. statistics_start_time is set to the current system date and time, the other counters are set to zero (0).

Permissions

Requires VIEW SERVER STATE permission.

See Also

Dynamic Management Views and Functions (Transact-SQL)
sys.dm_resource_governor_resource_pools (Transact-SQL)
sys.resource_governor_workload_groups (Transact-SQL)
ALTER RESOURCE GOVERNOR (Transact-SQL)

Security-Related Dynamic Management Views and Functions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management objects:

sys.dm_audit_actions (Transact-SQL)	sys.dm_cryptographic_provider_properties (Transact-SQL)
sys.dm_audit_class_type_map (Transact-SQL)	sys.dm_cryptographic_provider_sessions (Transact-SQL)
sys.dm_cryptographic_provider_algorithms (Transact-SQL)	sys.dm_database_encryption_keys (Transact-SQL)
sys.dm_cryptographic_provider_keys (Transact-SQL)	sys.dm_server_audit_status (Transact-SQL)

See Also

Extensible Key Management (EKM)
Transparent Data Encryption (TDE)
SQL Server Audit (Database Engine)

sys.dm_audit_actions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for every audit action that can be reported in the audit log and every audit action group that can be configured as part of SQL Server Audit. For more information about SQL Server Audit, see SQL Server Audit (Database Engine).

COLUMN NAME	DATA TYPE	DESCRIPTION
action_id	**varchar(4)**	ID of the audit action. Related to the **action_id** value written to each audit record. Is nullable. NULL for audit groups.
action_in_log	**bit**	Indicates whether an action can be written to an audit log. Values are as follows: 1 = Yes 0 = No
name	**sysname**	Name of the audit action or action group. Is not nullable.
class_desc	**nvarchar(120)**	The name of the class of the object that the audit action applies to. Can be any one of the Server, Database, or Schema scope objects, but does not include Schema objects. Is not nullable.
parent_class_desc	**nvarchar(120)**	Name of the parent class for the object described by class_desc. Is NULL if the class_desc is Server.
covering_parent_action_name	**nvarchar(120)**	Name of the audit action or audit group that contains the audit action described in this row. This is used to create a hierarchy of actions and covering actions. Is nullable.
configuration_level	**nvarchar(10)**	Indicates that the action or action group specified in this row is configurable at the Group or Action level. Is NULL if the action is not configurable.
containing_group_name	**nvarchar(120)**	The name of the audit group that contains the specified action. Is NULL if the value in name is a group.

Permissions

Principals must have **SELECT** permission. By default, this is granted to Public.

The visibility of the metadata in catalog views is limited to securables that a user either owns or on which the user has been granted some permission.. For more information, see Metadata Visibility Configuration.

See Also

CREATE SERVER AUDIT (Transact-SQL)
ALTER SERVER AUDIT (Transact-SQL)
DROP SERVER AUDIT (Transact-SQL)
CREATE SERVER AUDIT SPECIFICATION (Transact-SQL)
ALTER SERVER AUDIT SPECIFICATION (Transact-SQL)
DROP SERVER AUDIT SPECIFICATION (Transact-SQL)
CREATE DATABASE AUDIT SPECIFICATION (Transact-SQL)
ALTER DATABASE AUDIT SPECIFICATION (Transact-SQL)
DROP DATABASE AUDIT SPECIFICATION (Transact-SQL)
ALTER AUTHORIZATION (Transact-SQL)
sys.fn_get_audit_file (Transact-SQL)
sys.server_audits (Transact-SQL)
sys.server_file_audits (Transact-SQL)
sys.server_audit_specifications (Transact-SQL)
sys.server_audit_specification_details (Transact-SQL)
sys.database_audit_specifications (Transact-SQL)
sys.database_audit_specification_details (Transact-SQL)
sys.dm_server_audit_status (Transact-SQL)
sys.dm_audit_class_type_map (Transact-SQL)
Create a Server Audit and Server Audit Specification

sys.dm_audit_class_type_map (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a table that maps the class_type field in the audit log to the class_desc field in sys.dm_audit_actions. For more information about SQL Server Audit, see SQL Server Audit (Database Engine).

COLUMN NAME	DATA TYPE	DESCRIPTION
class_type	**char(2)**	The class type of the entity that was audited. Maps to the class_type written to the audit log and returned by the **get_audit_file()** function. Is not nullable.
class_type_desc	**nvarchar(120)**	The name for the auditable entity. Is not nullable.
securable_class_desc	**nvarchar(120)**	The securable object that maps to the class_type being audited. Is NULL if the class_type does not map to a securable object. Can be related to class_desc in sys.dm_audit_actions.

Permissions

Principal must have **SELECT** permission. By default, this is granted to Public.

See Also

CREATE SERVER AUDIT (Transact-SQL)
ALTER SERVER AUDIT (Transact-SQL)
DROP SERVER AUDIT (Transact-SQL)
CREATE SERVER AUDIT SPECIFICATION (Transact-SQL)
ALTER SERVER AUDIT SPECIFICATION (Transact-SQL)
DROP SERVER AUDIT SPECIFICATION (Transact-SQL)
CREATE DATABASE AUDIT SPECIFICATION (Transact-SQL)
ALTER DATABASE AUDIT SPECIFICATION (Transact-SQL)
DROP DATABASE AUDIT SPECIFICATION (Transact-SQL)
ALTER AUTHORIZATION (Transact-SQL)
sys.fn_get_audit_file (Transact-SQL)
sys.server_audits (Transact-SQL)
sys.server_file_audits (Transact-SQL)
sys.server_audit_specifications (Transact-SQL)
sys.server_audit_specification_details (Transact-SQL)
sys.database_audit_specifications (Transact-SQL)
sys.database_audit_specification_details (Transact-SQL)
sys.dm_server_audit_status (Transact-SQL)
sys.dm_audit_class_type_map

sys.dm_cryptographic_provider_algorithms (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the algorithms supported by an Extensible Key Management (EKM) provider.

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_cryptographic_provider_algorithms ( provider_id )
```

Arguments

provider_id
Identification number of the EKM provider, with no default.

Tables Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
algorithm_id	**int**	Is the identification number of the algorithm.
algorithm_tag	**nvarchar(60)**	Is the identification tag of the algorithm.
key_type	**nvarchar(128)**	Shows the key type. Returns either ASYMMETRIC KEY or SYMMETRIC KEY.
key_length	**int**	Indicates the key length in bits.

Permissions

The user must be a member of the public database role.

Examples

The following example shows the provider options for a provider with the identification number of `1234567`.

```
SELECT * FROM sys.dm_cryptographic_provider_algorithms(1234567);
GO
```

See Also

Extensible Key Management (EKM)

Security-Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_cryptographic_provider_keys (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the keys provided by a Extensible Key Management (EKM) provider.

Transact-SQL Syntax Conventions

Syntax

```
dm_cryptographic_provider_keys ( provider_id )
```

Arguments

provider_id
Identification number of the EKM provider, with no default.

Tables Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
key_id	**int**	Identification number of the key on the provider.
key_name	**nvarchar(512)**	Name of the key on the provider.
key_thumbprint	**varbinary(32)**	Thumbprint from the provider of the key.
algorithm_id	**int**	Identification number of the algorithm on the provider.
algorithm_tag	**int**	Tag of the algorithm on the provider.
key_type	**nchar(256)**	Type of key on the provider.
key_length	**int**	Length of the key on the provider.

Permissions

When this view is queried it will authenticate the user context with the provider and enumerate all keys visible to the user.

If the user cannot authenticate with the EKM provider, no key information will be returned.

Examples

The following example shows the key properties for a provider with the identification number of `1234567`.

```
SELECT * FROM sys.dm_cryptographic_provider_keys(1234567);
GO
```

See Also

Extensible Key Management (EKM)

sys.dm_cryptographic_provider_properties (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about registered cryptographic providers.

COLUMN NAME	DATA TYPE	DESCRIPTION
provider_id	**int**	Identification number of the cryptographic provider.
guid	**uniqueidentifier**	Unique provider GUID.
provider_version	**nvarchar(256)**	Version of the provider in the format '*aa.bb.cccc.dd*'.
sqlcrypt_version	**nvarchar(256)**	Major version of the SQL Server Cryptographic API in the format '*aa.bb.cccc.dd*'.
friendly_name	**nvarchar(2048)**	Name supplied by the provider.
authentication_type	**nvarchar(256)**	WINDOWS, BASIC, or OTHER.
symmetric_key_support	**tinyint**	0 (not supported) 1 (supported)
symmetric_key_export	**tinyint**	0 (not supported) 1 (supported)
symmetric_key_import	**tinyint**	0 (not supported) 1 (supported)
symmetric_key_persistance	**tinyint**	0 (not supported) 1 (supported)
asymmetric_key_support	**tinyint**	0 (not supported) 1 (supported)
asymmetric_key_export	**tinyint**	0 (not supported) 1 (supported)

COLUMN NAME	DATA TYPE	DESCRIPTION
symmetric_key_import	**tinyint**	0 (not supported) 1 (supported)
symmetric_key_persistance	**tinyint**	0 (not supported) 1 (supported)

Remarks

The sys.dm_cryptographic_provider_properties view is visible to the public.

See Also

Security Catalog Views (Transact-SQL)
Encryption Hierarchy
Extensible Key Management (EKM)
CREATE CRYPTOGRAPHIC PROVIDER (Transact-SQL)
Security-Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_cryptographic_provider_sessions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about open sessions for a cryptographic provider.

Syntax

```
sys.dm_cryptographic_provider_sessions(session_identifier)
```

Arguments

session_identifier
An integer indicating the sessions to be returned.

0 = Current connection only

1 = All cryptographic connections

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
provider_id	**int**	Identification number of the cryptographic provider.
session_handle	**varbytes(8)**	Cryptographic session handle.
identity	**nvarchar(128)**	Identity used to authenticate with the cryptographic provider.
spid	**short**	Session ID SPID of the connection. For more information, see @@SPID (Transact-SQL).

Remarks

The **sys.dm_cryptographic_provider_sessions** view is visible to the public for the current connection. To view all cryptographic connections, you must have the **CONTROL** server permission.

See Also

Security Catalog Views (Transact-SQL)
Extensible Key Management (EKM)
CREATE CRYPTOGRAPHIC PROVIDER (Transact-SQL)

sys.dm_database_encryption_keys (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the encryption state of a database and its associated database encryption keys. For more information about database encryption, see Transparent Data Encryption (TDE).

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	ID of the database.
encryption_state	**int**	Indicates whether the database is encrypted or not encrypted. 0 = No database encryption key present, no encryption 1 = Unencrypted 2 = Encryption in progress 3 = Encrypted 4 = Key change in progress 5 = Decryption in progress 6 = Protection change in progress (The certificate or asymmetric key that is encrypting the database encryption key is being changed.)
create_date	**datetime**	Displays the date the encryption key was created.
regenerate_date	**datetime**	Displays the date the encryption key was regenerated.
modify_date	**datetime**	Displays the date the encryption key was modified.
set_date	**datetime**	Displays the date the encryption key was applied to the database.
opened_date	**datetime**	Shows when the database key was last opened.
key_algorithm	**nvarchar(32)**	Displays the algorithm that is used for the key.
key_length	**int**	Displays the length of the key.

COLUMN NAME	DATA TYPE	DESCRIPTION
encryptor_thumbprint	**varbinary(20)**	Shows the thumbprint of the encryptor.
encryptor_type	**nvarchar(32)**	**Applies to**: SQL Server (SQL Server 2012 (11.x) through current version). Describes the encryptor.
percent_complete	**real**	Percent complete of the database encryption state change. This will be 0 if there is no state change.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

Security-Related Dynamic Management Views and Functions (Transact-SQL)
Transparent Data Encryption (TDE)
SQL Server Encryption
SQL Server and Database Encryption Keys (Database Engine)
Encryption Hierarchy
ALTER DATABASE SET Options (Transact-SQL)
CREATE DATABASE ENCRYPTION KEY (Transact-SQL)
ALTER DATABASE ENCRYPTION KEY (Transact-SQL)
DROP DATABASE ENCRYPTION KEY (Transact-SQL)

sys.dm_server_audit_status (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each server audit indicating the current state of the audit. For more information, see SQL Server Audit (Database Engine).

COLUMN NAME	DATA TYPE	DESCRIPTION
audit_id	**int**	ID of the audit. Maps to the **audit_id** field in the **sys.audits** catalog view.
name	**sysname**	Name of the audit. Same as the **name** field in the **sys.server_audits** catalog view.
status	**smallint**	Numeric status of the server audit: 0 = Not Started 1 = Started 2 = Runtime Fail 3 = Target Create Fail 4 = Shutting Down
status_desc	**nvarchar(256)**	String that shows the status of the server audit: NOT_STARTED STARTED RUNTIME_FAIL TARGET_CREATION_FAILED SHUTTING_DOWN
status_time	**datetime2**	Timestamp in UTC of the last status change for the audit.
event_session_address	**varbinary(8)**	Address of the Extended Events session associated with the audit. Related to the **sys.db_xe_sessions.address** catalog view.
audit_file_path	**nvarchar(256)**	Full path and file name of the audit file target that is currently being used. Only populated for file audits.

COLUMN NAME	DATA TYPE	DESCRIPTION
audit_file_size	**bigint**	Approximate size of the audit file, in bytes. Only populated for file audits.

Permissions

Principals must have **VIEW SERVER STATE** and **SELECT** permissions.

The visibility of the metadata in catalog views is limited to securables that a user either owns or on which the user has been granted some permission. For more information, see Metadata Visibility Configuration.

See Also

CREATE SERVER AUDIT (Transact-SQL)
ALTER SERVER AUDIT (Transact-SQL)
DROP SERVER AUDIT (Transact-SQL)
CREATE SERVER AUDIT SPECIFICATION (Transact-SQL)
ALTER SERVER AUDIT SPECIFICATION (Transact-SQL)
DROP SERVER AUDIT SPECIFICATION (Transact-SQL)
CREATE DATABASE AUDIT SPECIFICATION (Transact-SQL)
ALTER DATABASE AUDIT SPECIFICATION (Transact-SQL)
DROP DATABASE AUDIT SPECIFICATION (Transact-SQL)
ALTER AUTHORIZATION (Transact-SQL)
sys.fn_get_audit_file (Transact-SQL)
sys.server_audits (Transact-SQL)
sys.server_file_audits (Transact-SQL)
sys.server_audit_specifications (Transact-SQL)
sys.server_audit_specification_details (Transact-SQL)
sys.database_audit_specifications (Transact-SQL)
sys.database_audit_specification_details (Transact-SQL)
sys.dm_server_audit_status
sys.dm_audit_actions (Transact-SQL)
sys.dm_audit_class_type_map (Transact-SQL)
Create a Server Audit and Server Audit Specification

Server-Related Dynamic Management Views and Functions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the dynamic management views that are associated with the SQL Server, Full-text, and SQL Server Agent services that are installed on the host server. You can use these views to return property information for these services. These views also contain configuration, installation, and memory dump file information.

In This Section

sys.dm_server_memory_dumps	sys.dm_server_services
sys.dm_server_registry	

sys.dm_server_memory_dumps (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row for each memory dump file generated by the SQL Server Database Engine. Use this dynamic management view to troubleshoot potential issues.

COLUMN NAME	DATA TYPE	DESCRIPTION
filename	**nvarchar(256)**	Path and name of the memory dump file. Cannot be null.
creation_time	**datetimeoffset(7)**	Date and time the file was created. Cannot be null.
size_in_bytes	**bigint**	Size (in bytes) of the file. Is nullable.

General Remarks

The dump type may be a minidump, all-thread dump, or a full dump. The files have an extension of .mdmp.

Security

Dump files might contain sensitive information. To help protect sensitive information, you can use an access control list (ACL) to restrict access to the files, or copy the files to a folder that has restricted access. For example, before you send your debug files to Microsoft support services, we recommend that you remove any sensitive or confidential information.

Permissions

Requires VIEW SERVER STATE permission.

sys.dm_server_services (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the SQL Server, Full-Text, and SQL Server Agent services in the current instance of SQL Server. Use this dynamic management view to report status information about these services.

COLUMN NAME	DATA TYPE	DESCRIPTION
servicename	**nvarchar(256)**	Name of the SQL Server Database Engine, Full-text, or SQL Server Agent service. Cannot be null.
startup_type	**int**	Indicates the start mode of the service. The following are the possible values and their corresponding descriptions. 0: Other 1: Other 2: Automatic 3: Manual 4: Disabled Is nullable.
startup_desc	**nvarchar(256)**	Describes the start mode of the service. The following are the possible values and their corresponding descriptions. Other: Other (boot start) Other: Other (system start) Automatic: Auto start Manual: Demand start Disabled: Disabled Cannot be null.
status	**int**	Indicates the current status of the service. The following are the possible values and their corresponding descriptions. 1: Stopped 2: Other (start pending) 3: Other (stop pending) 4: Running 5: Other (continue pending) 6: Other (pause pending) 7: Paused Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
status_desc	**nvarchar(256)**	Describes the current status of the service. The following are the possible values and their corresponding descriptions. Stopped: The service is stopped. Other (start operation pending): The service is in the process of starting. Other (stop operation pending): The service is in the process of stopping. Running: The service is running. Other (continue operations pending): The service is in a pending state. Other (pause pending): The service is in the process of pausing. Paused: The service is paused. Cannot be null.
process_id	**int**	The process ID of the service. Cannot be null.
last_startup_time	**datetimeoffset(7)**	The date and time the service was last started. Is nullable.
service_account	**nvarchar(256)**	The account authorized to control the service. This account can start or stop the service, or modify service properties. Cannot be null.
filename	**nvarchar(256)**	The path and filename of the service executable. Cannot be null.
is_clustered	**nvarchar(1)**	Indicates whether the service is installed as a resource of a clustered server. Cannot be null.
cluster_nodename	**nvarchar(256)**	The name of the cluster node on which the service is installed. Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
instant_file_initialization_enabled	**nvarchar(1)**	Specifies whether instant file initialization is enabled for the SQL Server Database Engine service. Y = instant file initialization is enabled for the service. N = instant file initialization is disabled for the service. Is nullable. **Note:** Does not apply to other services such as the SQL Server Agent. **Applies to:** SQL Server (Starting with SQL Server 2012 (11.x) SP4, and SQL Server 2016 (13.x) SP1 through through SQL Server 2017).

Security

Permissions

Requires `VIEW SERVER STATE` permission on the server.

See Also

sys.dm_server_registry (Transact-SQL)

sys.dm_server_registry (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns configuration and installation information that is stored in the Windows registry for the current instance of SQL Server. Returns one row per registry key. Use this dynamic management view to return information such as the SQL Server services that are available on the host machine or network configuration values for the instance of SQL Server.

COLUMN NAME	DATA TYPE	DESCRIPTION
registry_key	**nvarchar(256)**	Registry key name. Is nullable.
value_name	**nvarchar(256)**	Key value name. This is the item shown in the **Name** column of the Registry Editor. Is nullable.
value_data	**sql_variant**	Value of the key data. This is the value shown in the **Data** column of the Registry Editor for a given entry. Is nullable.

Security

Permissions

Requires VIEW SERVER STATE permission on the server.

Examples

A. Display the SQL Server services

The following example returns registry key values for the SQL Server and SQL Server Agent services for the current instance of SQL Server.

```
SELECT registry_key, value_name, value_data
FROM sys.dm_server_registry
WHERE registry_key LIKE N'%ControlSet%';
```

B. Display the SQL Server Agent registry key values

The following example returns the SQL Server Agent registry key values for the current instance of SQL Server.

```
SELECT registry_key, value_name, value_data
FROM sys.dm_server_registry
WHERE registry_key LIKE N'%SQLAgent%';
```

C. Display the current version of the instance of SQL Server

The following example returns the version of the current instance of SQL Server.

```sql
SELECT registry_key, value_name, value_data
FROM sys.dm_server_registry
WHERE registry_key = N'CurrentVersion';
```

D. Display the parameters passed to the instance of SQL Server during startup

The following example returns the parameters that are passed to the instance of SQL Server during startup.

```sql
SELECT registry_key, value_name, value_data
FROM sys.dm_server_registry
WHERE registry_key LIKE N'%Parameters';
```

E. Return network configuration information for the instance of SQL Server

The following example returns network configuration values for the current instance of SQL Server.

```sql
SELECT registry_key, value_name, value_data
FROM sys.dm_server_registry
WHERE registry_key LIKE N'%SuperSocketNetLib%';
```

See Also

sys.dm_server_services (Transact-SQL)

Service Broker Related Dynamic Management Views (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management views.

sys.dm_broker_activated_tasks	sys.dm_broker_connections
sys.dm_broker_forwarded_messages	sys.dm_broker_queue_monitors

See Also

Dynamic Management Views and Functions (Transact-SQL)
System Views (Transact-SQL)

sys.dm_broker_activated_tasks (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each stored procedure activated by Service Broker.

COLUMN NAME	DATA TYPE	DESCRIPTION
spid	**int**	ID of the session of the activated stored procedure. NULLABLE.
database_id	**smallint**	ID of the database in which the queue is defined. NULLABLE.
queue_id	**int**	ID of the object of the queue for which the stored procedure was activated. NULLABLE.
procedure_name	**nvarchar(650)**	Name of the activated stored procedure. NULLABLE.
execute_as	**int**	ID of the user that the stored procedure runs as. NULLABLE.

Permissions

Requires VIEW SERVER STATE permission on the server.

Physical Joins

Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_broker_activated_tasks.spid	dm_exec_sessions.session_id	One-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)
Service Broker Related Dynamic Management Views (Transact-SQL)

sys.dm_broker_connections (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each Service Broker network connection. The following table provides more information:

COLUMN NAME	DATA TYPE	DESCRIPTION
connection_id	**uniqueidentifier**	Identifier of the connection. NULLABLE.
transport_stream_id	**uniqueidentifier**	Identifier of the SQL Server Network Interface (SNI) connection used by this connection for TCP/IP communications. NULLABLE.
state	**smallint**	Current state of the connection. NULLABLE. Possible values: 1 = NEW 2 = CONNECTING 3 = CONNECTED 4 = LOGGED_IN 5 = CLOSED
state_desc	**nvarchar(60)**	Current state of the connection. NULLABLE. Possible values: NEW CONNECTING CONNECTED LOGGED_IN CLOSED
connect_time	**datetime**	Date and time at which the connection was opened. NULLABLE.
login_time	**datetime**	Date and time at which login for the connection succeeded. NULLABLE.
authentication_method	**nvarchar(128)**	Name of the Windows Authentication method, such as NTLM or KERBEROS. The value comes from Windows. NULLABLE.

COLUMN NAME	DATA TYPE	DESCRIPTION
principal_name	**nvarchar(128)**	Name of the login that was validated for connection permissions. For Windows Authentication, this value is the remote user name. For certificate authentication, this value is the certificate owner. NULLABLE.
remote_user_name	**nvarchar(128)**	Name of the peer user from the other database that is used by Windows Authentication. NULLABLE.
last_activity_time	**datetime**	Date and time at which the connection was last used to send or receive information. NULLABLE.
is_accept	**bit**	Indicates whether the connection originated on the remote side. NULLABLE. 1 = The connection is a request accepted from the remote instance. 0 = The connection was started by the local instance.
login_state	**smallint**	State of the login process for this connection. Possible values: 0 = INITIAL 1 = WAIT LOGIN NEGOTIATE 2 = ONE ISC 3 = ONE ASC 4 = TWO ISC 5 = TWO ASC 6 = WAIT ISC Confirm 7 = WAIT ASC Confirm 8 = WAIT REJECT 9 = WAIT PRE-MASTER SECRET 10 = WAIT VALIDATION 11 = WAIT ARBITRATION 12 = ONLINE 13 = ERROR

COLUMN NAME	DATA TYPE	DESCRIPTION
login_state_desc	**nvarchar(60)**	Current state of login from the remote computer. Possible values: Connection handshake is initializing. Connection handshake is waiting for Login Negotiate message. Connection handshake has initialized and sent security context for authentication. Connection handshake has received and accepted security context for authentication. Connection handshake has initialized and sent security context for authentication. There is an optional mechanism available for authenticating the peers. Connection handshake has received and sent accepted security context for authentication. There is an optional mechanism available for authenticating the peers. Connection handshake is waiting for Initialize Security Context Confirmation message. Connection handshake is waiting for Accept Security Context Confirmation message. Connection handshake is waiting for SSPI rejection message for failed authentication. Connection handshake is waiting for Pre-Master Secret message. Connection handshake is waiting for Validation message. Connection handshake is waiting for Arbitration message. Connection handshake is complete and is online (ready) for message exchange. Connection is in error.
peer_certificate_id	**int**	The local object ID of the certificate that is used by the remote instance for authentication. The owner of this certificate must have CONNECT permissions to the Service Broker endpoint. NULLABLE.

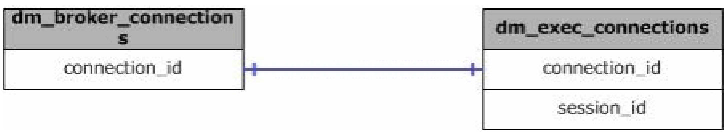
COLUMN NAME	DATA TYPE	DESCRIPTION
encryption_algorithm	**smallint**	Encryption algorithm that is used for this connection. NULLABLE. Possible values: **Value | Description | Corresponding DDL option** 0 | none | Disabled 1 | SIGNING ONLY 2 | AES , RC4 | Required | Required algorithm RC4} 3 | AES |Required algorithm AES **Note:** The RC4 algorithm is only supported for backward compatibility. New material can only be encrypted using RC4 or RC4_128 when the database is in compatibility level 90 or 100. (Not recommended.) Use a newer algorithm such as one of the AES algorithms instead. In SQL Server 2012 (11.x) and later versions, material encrypted using RC4 or RC4_128 can be decrypted in any compatibility level.
encryption_algorithm_desc	**nvarchar(60)**	Textual representation of the encryption algorithm. NULLABLE. Possible Values: **Description | Corresponding DDL option** NONE | Disabled RC4 | {Required | Required Algorithm RC4} AES | Required Algorithm AES NONE, RC4 | {Supported | Supported Algorithm RC4} NONE, AES | Supported Algorithm RC4 RC4, AES | Required Algorithm RC4 AES AES, RC4 | Required Algorithm AES RC4 NONE, RC4, AES | Supported Algorithm RC4 AES NONE, AES, RC4 | Supported Algorithm AES RC4
receives_posted	**smallint**	Number of asynchronous network receives that have not yet completed for this connection. NULLABLE.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_receive_flow_controlled	**bit**	Whether network receives have been postponed due to flow control because the network is busy. NULLABLE. 1 = True
sends_posted	**smallint**	The number of asynchronous network sends that have not yet completed for this connection. NULLABLE.
is_send_flow_controlled	**bit**	Whether network sends have been postponed due to network flow control because the network is busy. NULLABLE. 1 = True
total_bytes_sent	**bigint**	Total number of bytes that were sent by this connection. NULLABLE.
total_bytes_received	**bigint**	Total number of bytes that were received by this connection. NULLABLE.
total_fragments_sent	**bigint**	Total number of Service Broker message fragments that were sent by this connection. NULLABLE.
total_fragments_received	**bigint**	Total number of Service Broker message fragments that were received by this connection. NULLABLE.
total_sends	**bigint**	Total number of network send requests that were issued by this connection. NULLABLE.
total_receives	**bigint**	Total number of network receive requests that were issued by this connection. NULLABLE.
peer_arbitration_id	**uniqueidentifier**	Internal identifier for the endpoint. NULLABLE.

Permissions

Requires VIEW SERVER STATE permission on the server.

Physical Joins



Relationship Cardinalities

FROM	TO	RELATIONSHIP
dm_broker_connections.connection_id	**dm_exec_connections.connection_id**	One-to-one

See Also

Dynamic Management Views and Functions (Transact-SQL)
Service Broker Related Dynamic Management Views (Transact-SQL)

sys.dm_broker_forwarded_messages (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each Service Broker message that an instance of SQL Server is in the process of forwarding.

COLUMN NAME	DATA TYPE	DESCRIPTION
conversation_id	**uniqueidentifier**	ID of the conversation to which this message belongs. NULLABLE.
is_initiator	**bit**	Indicates whether this message is from the initiator of the conversation. NULLABLE. 0 = Not from initiator 1 = From initiator
to_service_name	**nvarchar(512)**	Name of the service to which this message is sent. NULLABLE.
to_broker_instance	**nvarchar(512)**	Identifier of the broker that hosts the service to which this message is sent. NULLABLE.
from_service_name	**nvarchar(512)**	Name of the service that this message is from. NULLABLE.
from_broker_instance	**nvarchar(512)**	Identifier of the broker that hosts the service that this message is from. NULLABLE.
adjacent_broker_address	**nvarchar(512)**	Network address to which this message is being sent. NULLABLE.
message_sequence_number	**bigint**	Sequence number of the message in the dialog box. NULLABLE.
message_fragment_number	**int**	If the dialog message is fragmented, this is the fragment number that this transport message contains. NULLABLE.
hops_remaining	**tinyint**	Number of times the message may be retransmitted before reaching the final destination. Every time the message is forwarded, this number decreases by 1. NULLABLE.
time_to_live	**int**	Maximum time for the message to remain active. When this reaches 0, the message is discarded. NULLABLE.

COLUMN NAME	DATA TYPE	DESCRIPTION
time_consumed	**int**	Time that the message has already been active. Every time the message is forwarded, this number is increased by the time it has taken to forward the message. Not NULLABLE.
message_id	**uniqueidentifier**	ID of the message. NULLABLE.

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Service Broker Related Dynamic Management Views (Transact-SQL)

sys.dm_broker_queue_monitors (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each queue monitor in the instance. A queue monitor manages activation for a queue.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	Object identifier for the database that contains the queue that the monitor watches. NULLABLE.
queue_id	**int**	Object identifier for the queue that the monitor watches. NULLABLE.
state	**nvarchar(32)**	State of the monitor. NULLABLE. This is one of the following: **INACTIVE** **NOTIFIED** **RECEIVES_OCCURRING**
last_empty_rowset_time	**datetime**	Last time that a RECEIVE from the queue returned an empty result. NULLABLE.
last_activated_time	**datetime**	Last time that this queue monitor activated a stored procedure. NULLABLE.
tasks_waiting	**int**	Number of sessions that are currently waiting within a RECEIVE statement for this queue. NULLABLE. Note: This number includes any session executing a receive statement, regardless of whether the queue monitor started the session. This is if you use WAITFOR together with RECEIVE. Basically, these tasks are waiting for messages to arrive on the queue.

Permissions

Requires VIEW SERVER STATE permission on the server.

Examples

A. Current status queue monitor

This scenario provides the current status of all message queues.

```
SELECT t1.name AS [Service_Name],  t3.name AS [Schema_Name],  t2.name AS [Queue_Name],
CASE WHEN t4.state IS NULL THEN 'Not available'
ELSE t4.state
END AS [Queue_State],
CASE WHEN t4.tasks_waiting IS NULL THEN '--'
ELSE CONVERT(VARCHAR, t4.tasks_waiting)
END AS tasks_waiting,
CASE WHEN t4.last_activated_time IS NULL THEN '--'
ELSE CONVERT(varchar, t4.last_activated_time)
END AS last_activated_time ,
CASE WHEN t4.last_empty_rowset_time IS NULL THEN '--'
ELSE CONVERT(varchar,t4.last_empty_rowset_time)
END AS last_empty_rowset_time,
(
SELECT COUNT(*)
FROM sys.transmission_queue t6
WHERE (t6.from_service_name = t1.name) ) AS [Tran_Message_Count]
FROM sys.services t1    INNER JOIN sys.service_queues t2
ON ( t1.service_queue_id = t2.object_id )
INNER JOIN sys.schemas t3 ON ( t2.schema_id = t3.schema_id )
LEFT OUTER JOIN sys.dm_broker_queue_monitors t4
ON ( t2.object_id = t4.queue_id  AND t4.database_id = DB_ID() )
INNER JOIN sys.databases t5 ON ( t5.database_id = DB_ID() );
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Service Broker Related Dynamic Management Views (Transact-SQL)

Spatial Data - sys.dm_db_objects_disabled_on_compatibility_level_change

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Lists the indexes and constraints that will be disabled as a result of changing compatibility level in SQL Server. Indexes and constraints that contain persisted computed columns whose expressions use spatial UDTs will be disabled after upgrading or changing compatibility level. Use this dynamic management function to determine the impact of a change in compatibility level.

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_db_objects_disabled_on_compatibility_level_change ( compatibility_level )
```

Arguments

compatibility_level
int that identifies the compatibility level that you plan to set.

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
class	**int**	1 = constraints 7 = indexes and heaps
class_desc	**nvarchar(60)**	OBJECT or COLUMN for constraints INDEX for indexes and heaps
major_id	**int**	OBJECT ID of constraints OBJECT ID of table that contains indexes and heaps
minor_id	**int**	NULL for constraints Index_id for indexes and heaps

COLUMN NAME	DATA TYPE	DESCRIPTION
dependency	**nvarchar(60)**	Description of the dependency that is causing the constraint or index to be disabled. The same values are also used in the warnings that are raised during upgrade. Examples include the following: "space" for an intrinsic "geometry" for a system UDT "geography::Parse" for a method of a system UDT

General Remarks

Persisted computed columns that use some intrinsic functions are disabled when the compatibility level is changed. Also, persisted computed columns that use any Geometry or Geography method are disabled when a database is upgraded.

Which functions cause persisted computed columns to be disabled?

When the following functions are used in the expression of a persisted computed column, they cause indexes and constraints that reference those columns to be disabled when the compatibility level is changed from 80 to 90:

- **IsNumeric**

  When the following functions are used in the expression of a persisted computed column, they cause indexes and constraints that reference those columns to be disabled when the compatibility level is changed from 100 to 110 or higher:

- **Soundex**
- **Geography:: GeomFromGML**
- **Geography:: STGeomFromText**
- **Geography:: STLineFromText**
- **Geography:: STPolyFromText**
- **Geography:: STMPointFromText**
- **Geography:: STMLineFromText**
- **Geography:: STMPolyFromText**
- **Geography:: STGeomCollFromText**
- **Geography:: STGeomFromWKB**
- **Geography:: STLineFromWKB**
- **Geography:: STPolyFromWKB**
- **Geography:: STMPointFromWKB**
- **Geography:: STMLineFromWKB**
- **Geography:: STMPolyFromWKB**
- **Geography:: STUnion**
- **Geography:: STIntersection**

- **Geography:: STDifference**
- **Geography:: STSymDifference**
- **Geography:: STBuffer**
- **Geography:: BufferWithTolerance**
- **Geography:: Parse**
- **Geography:: Reduce**

Behavior of the disabled objects

Indexes

If the clustered index is disabled, or if a non-clustered index is forced, the following error is raised: "The query processor is unable to produce a plan because the index '%.*ls' on table or view '%.*ls' is disabled." To re-enable these objects, rebuild the indexes after upgrade by calling **ALTER INDEX ON ... REBUILD**.

Heaps

If a table with a disabled heap is used, the following error is raised. To re-enable these objects, rebuild after upgrade by calling **ALTER INDEX ALL ON ... REBUILD**.

```
// ErrorNumber: 8674
// ErrorSeverity: EX_USER
// ErrorFormat: The query processor is unable to produce a plan because the table or view '%.*ls' is disabled.
// ErrorCause: The table has a disabled heap.
// ErrorCorrectiveAction: Rebuild the disabled heap to enable it.
// ErrorInserts: table or view name
// ErrorOwner: mtintor
// ErrorFirstProduct: SQL11
```

If you try to rebuild the heap during an online operation, an error is raised.

Check Constraints and Foreign Keys

Disabled check constraints and foreign keys do not raise an error. However, the constraints are not enforced when rows are modified. To re-enable these objects, check the constraints after upgrading by calling **ALTER TABLE ... CHECK CONSTRAINT**.

Persisted Computed Columns

Since it is not possible to disable a single column, the entire table is disabled by disabling the clustered index or heap.

Security

Permissions

Requires the VIEW DATABASE STATE permission.

Example

The following example shows a query on **sys.dm_db_objects_disabled_on_compatibility_level_change** to find the objects impacted by changing the compatibility level to 120.

```
SELECT * FROM sys.dm_db_objects_disabled_on_compatibility_level_change(120);
GO
```

SQL and Parallel Data Warehouse Dynamic Management Views

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This topic lists the SQL Data Warehouse and Parallel Data Warehouse dynamic management views (DMVs).

All SQL Data Warehouse and Parallel Data Warehouse DMVs begin with **sys.dm_pdw**.

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views

The following dynamic management views apply to both SQL Data Warehouse and Parallel Data Warehouse:

sys.dm_pdw_dms_cores (Transact-SQL)

sys.dm_pdw_dms_external_work (Transact-SQL)

sys.dm_pdw_dms_workers (Transact-SQL)

sys.dm_pdw_errors (Transact-SQL)

sys.dm_pdw_exec_connections (Transact-SQL)

sys.dm_pdw_exec_requests (Transact-SQL)

sys.dm_pdw_exec_sessions (Transact-SQL)

sys.dm_pdw_hadoop_operations (Transact-SQL)

sys.dm_pdw_lock_waits (Transact-SQL)

sys.dm_pdw_nodes (Transact-SQL)

sys.dm_pdw_nodes_database_encryption_keys (Transact-SQL)

sys.dm_pdw_os_threads (Transact-SQL)

sys.dm_pdw_request_steps (Transact-SQL)

sys.dm_pdw_resource_waits (Transact-SQL)

sys.dm_pdw_sql_requests (Transact-SQL)

sys.dm_pdw_sys_info (Transact-SQL)

sys.dm_pdw_wait_stats (Transact-SQL)

sys.dm_pdw_waits (Transact-SQL)

Parallel Data Warehouse Dynamic Management Views

The following dynamic management views apply to Parallel Data Warehouse only:

sys.dm_pdw_component_health_active_alerts (Transact-SQL) sys.dm_pdw_component_health_alerts (Transact-SQL)

sys.dm_pdw_component_health_status (Transact-SQL)

sys.dm_pdw_diag_processing_stats (Transact-SQL)

sys.dm_pdw_network_credentials (Transact-SQL)

sys.dm_pdw_node_status (Transact-SQL)

sys.dm_pdw_os_event_logs (Transact-SQL)

sys.dm_pdw_os_performance_counters (Transact-SQL)

sys.dm_pdw_query_stats_xe (Transact-SQL)

sys.dm_pdw_query_stats_xe_file (Transact-SQL)

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_pdw_component_health_active_alerts (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Stores active alerts on Parallel Data Warehouse components.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
pdw_node_id	**int**	Unique identifier of a Parallel Data Warehouse node. pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL
component_id	**int**	The ID of the component. See sys.pdw_health_components (Transact-SQL). pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL
component_instance_id	**nvarchar(255)**	pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL
alert_id	**int**	The ID for the alert type. See sys.pdw_health_alerts (Transact-SQL). pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL
alert_instance_id	**nvarchar(36)**	Identifies an instance of a given alert. pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
current_value	**nvarchar(255)**	Used when the alert is of type StatusChange. This is the current component status. Value is NULL for alerts of type Threshold. See sys.pdw_health_alerts (Transact-SQL) for a list of alert types.	NULL
previous_value	**nvarchar(255)**	Used when the alert is of type StatusChange. This is the previous component status. Value is NULL for alerts of type Threshold. See sys.pdw_health_alerts (Transact-SQL) for a list of alert types.	NULL
create_time	**datetime**	Time and date when the alert was generated.	NOT NULL

For information about the maximum rows retained by this view, see "Minimum and Maximum Values" in the Parallel Data Warehouse product documentation.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_component_health_alerts (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Stores previously issued alerts on appliance components.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
pdw_node_id	**int**	Unique identifier of a Parallel Data Warehouse node. pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL
component_id	**int**	The ID of the component. See sys.pdw_health_components (Transact-SQL). pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL
component_instance_id	**nvarchar(255)**	pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL
alert_id	**int**	The ID for the alert type. See sys.pdw_health_alerts (Transact-SQL). pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL
alert_instance_id	**nvarchar(36)**	Identifies an instance of a given alert. pdw_node_id, component_id, component_instance_id, alert_id, and alert_instance_id form the key for this view.	NOT NULL

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
previous_value	**nvarchar(255)**	Used when the alert is of type StatusChange. This is the previous component status. Value is NULL for alerts of type Threshold. See sys.pdw_health_alerts (Transact-SQL) for a list of alert types.	NULL
current_value	**nvarchar(255)**	Used when the alert is of type StatusChange. This is the current component status. Value is NULL for alerts of type Threshold. See sys.pdw_health_alerts (Transact-SQL) for a list of alert types.	NULL
create_time	**datetime**	Time and date when the alert was generated.	NOT NULL

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_component_health_status (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about the current health of appliance components.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
pdw_node_id	**int**		Not NULL
component_id	int	The ID of the component. See sys.pdw_health_components (Transact-SQL). pdw_node_id, component_id, property_id, and component_instance_id form the key for this view.	Not NULL
property_id	**int**	The ID of the property. See sys.pdw_health_component_properties (Transact-SQL).	NOT NULL
component_instance_id	**nvarchar(255)**	Identifies an instance of a component. For example, an instance of a CPU might be identified by component_instance_id='CPU1'. pdw_node_id, component_id, property_id, and component_instance_id form the key for this view.	NOT NULL
property_value	**nvarchar(255)**	The current property value.	NULL
update_time	**datetime**	The last time the metric was updated.	NOT NULL

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_diag_processing_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Displays information related to all internal diagnostic events that could be incorporated into diagnostic sessions defined by the administrator. Query this view to understand the statistics behind the diagnostics and eventing subsystems that drive the population of all the other DMVs. There are a group of queues for each process on each node.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	Appliance node this log is from.
process_id	**int**	Identifier of the process running submitting this statistic.
target_name	**nvarchar(255)**	The name of the queue.
queue_size	**int**	The number of items in the process queue. The queue size is usually 0. A positive number indicates that the system is under stress and is building backlog of events. A positive count in the other columns means system has become corrupted for that particular queue and any related DMVs.
lost_events_count	**bigint**	The number of events lost.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_dms_cores (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all DMS services running on the Compute nodes of the appliance. It lists one row per service instance, which is currently one row per node.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
dms_core_id	**int**	Unique numeric id associated with this DMS core. Key for this view.	Set to the pdw_node_id of the node that this DMS core is running on.
pdw_node_id	**int**	ID of the node on which this DMS service is running.	See node_id in sys.dm_pdw_nodes (Transact-SQL).
status	**nvarchar(32)**	Current status of the DMS service.	Information not available.

For information about the maximum rows retained by this view, see the Maximum System View Values section in the Minimum and Maximum Values (SQL Server PDW) topic.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_dms_external_work (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Data Warehouse system view that holds information about all Data Movement Service (DMS) steps for external operations.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
request_id	**nvarchar(32)**	Query that is using this DMS worker. request_id, step_index, and dms_step_index form the key for this view.	Same as request_id in sys.dm_pdw_exec_requests (Transact-SQL).
step_index	**int**	Query step that is invoking this DMS worker. request_id, step_index, and dms_step_index form the key for this view.	Same as step_index in sys.dm_pdw_request_steps (Transact-SQL).
dms_step_index	**int**	Current step in the DMS plan. request_id, step_index, and dms_step_index form the key for this view.	Same as dms__step_index in sys.dm_pdw_dms_workers (Transact-SQL).
pdw_node_id	**int**	Node that is running the DMS worker.	Same as node_id in sys.dm_pdw_nodes (Transact-SQL).
type	**nvarchar(60)**	Type of external operation this node is running. FILE SPLIT is an operation on an external Hadoop file that has been split into multiple smaller falls.	'FILE SPLIT'
work_id	**int**	The file split ID.	Greater than or equal to 0. Unique per Compute node.
input_name	**nvarchar(60)**	String name for the input being read.	For a Hadoop file, this is the Hadoop file name.
read_location	**bigint**	Offset of read location.	
estimated_bytes_processed	**bigint**	Number of bytes processed by this worker.	Greater than or equal to 0.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
length	**bigint**	Number of bytes in the file split. For Hadoop, this is the size of the HDFS block.	User-defined. The default is 64 MB.
status	**nvarchar(32)**	State of the worker.	Pending, Processing, Done, Failed, Aborted
start_time	**datetime**	Time at which execution of this worker started.	Greater than or equal to start time of the query step this worker belongs to. See sys.dm_pdw_request_steps (Transact-SQL).
end_time	**datetime**	Time at which execution ended, failed, or was cancelled.	NULL for ongoing or queued workers. Otherwise, greater than start_time.
total_elapsed_time	**int**	Total time spent in execution, in milliseconds.	Greater than or equal to 0. If total_elapsed_time exceeds the maximum value for an integer, total_elapsed_time will continue to be the maximum value. This condition will generate the warning "The maximum value has been exceeded." The maximum value in milliseconds is equivalent to 24.8 days.

For information about the maximum rows retained by this view, see Maximum System View Values.

See Also

System Views (Transact-SQL)

sys.dm_pdw_dms_workers (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all workers completing DMS steps.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
request_id	**nvarchar(32)**	Query that this DMS worker is part of. request_id, step_index, and dms_step_index form the key for this view.	See request_id in sys.dm_pdw_exec_requests (Transact-SQL).
step_index	**int**	Query step this DMS worker is part of. request_id, step_index, and dms_step_index form the key for this view.	See step_index in sys.dm_pdw_request_steps (Transact-SQL).
dms_step_index	**int**	Step in the DMS plan that this worker is running. request_id, step_index, and dms_step_index form the key for this view.	
pdw_node_id	**int**	Node that the worker is running on.	See node_id in sys.dm_pdw_nodes (Transact-SQL).
distribution_id	**Int**	Distribution that the worker is running on, if any.	See distribution_id in sys.pdw_distributions (Transact-SQL).
type	**nvarchar(32)**	Type of DMS worker thread this entry represents.	'DIRECT_CONVERTER', 'DIRECT_READER', 'FILE_READER', 'HASH_CONVERTER', 'HASH_READER', 'ROUNDROBIN_CONVERTER', 'EXPORT_READER', 'EXTERNAL_READER', 'EXTERNAL_WRITER', 'PARALLEL_COPY_READER', 'REJECT_WRITER', 'WRITER'
status	**nvarchar(32)**	Status of the DMS worker.	Information not available.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
bytes_per_sec	**bigint**	Read or write throughput in the last second.	Greater than or equal to 0. NULL if the query was cancelled or failed before the worker could execute.
bytes_processed	**bigint**	Total bytes processed by this worker.	Greater than or equal to 0. NULL if the query was cancelled or failed before the worker could execute.
rows_processed	**bigint**	Number of rows read or written for this worker.	Greater than or equal to 0. NULL if the query was cancelled or failed before the worker could execute.
start_time	**datetime**	Time at which execution of this worker started.	Greater than or equal to start time of the query step this worker belongs to. See sys.dm_pdw_request_steps (Transact-SQL).
end_time	**datetime**	Time at which execution ended, failed, or was cancelled.	NULL for ongoing or queued workers. Otherwise, greater than start_time.
total_elapsed_time	**int**	Total time spent in execution, in milliseconds.	Greater than or equal to 0. Total time elapsed since system start or restart. If total_elapsed_time exceeds the maximum value for an integer (24.8 days in milliseconds), it will cause materialization failure due to overflow. The maximum value in milliseconds is equivalent to 24.8 days.
cpu_time	**bigint**	CPU time consumed by this worker, in milliseconds.	Greater than or equal to 0.
query_time	**int**	Period of time before SQL starts returning rows to the thread, in milliseconds.	Greater than or equal to 0.
buffers_available	**int**	Number of unused buffers.	NULL if the query was cancelled or failed before the worker could execute.
sql_spid	**int**	Session id on the SQL Server instance performing the work for this DMS worker.	
dms_cpid	**int**	Process ID of the actual thread running.	

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
error_id	**nvarchar(36)**	Unique identifier of the error that occurred during execution of this worker, if any.	See error_id in sys.dm_pdw_request_steps (Transact-SQL).
source_info	**nvarchar(4000)**	For a reader, specification of the source tables and columns.	
destination_info	**nvarchar(4000)**	For a writer, specification of the destination tables.	

For information about the maximum rows retained by this view, see Maximum System View Values.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_errors (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all errors encountered during execution of a request or query.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
error_id	**nvarchar(36)**	Key for this view. Unique numeric id associated with the error.	Unique across all query errors in the system.
source	**nvarchar(64)**	Information not available.	Information not available.
type	**nvarchar(4000)**	Type of error that occurred.	Information not available.
create_time	**datetime**	Time at which the error occurred.	Smaller or equal to current time.
pwd_node_id	**int**	Identifier of the specific node involved, if any. For additional information on node ids, see sys.dm_pdw_nodes (Transact-SQL).	
session_id	**nvarchar(32)**	Identifier of the session involved, if any. For additional information on session ids, see sys.dm_pdw_exec_sessions (Transact-SQL).	
request_id	**nvarchar(32)**	Identifier of the request involved, if any. For additional information on request ids, see sys.dm_pdw_exec_requests (Transact-SQL).	
spid	**int**	spid of the SQL Server session involved, if any.	
thread_id	**int**	Information not available.	
details	**nvarchar(4000)**	Holds the full error text description.	

For information about the maximum rows retained by this view, see the Maximum System View Values section in the Minimum and Maximum Values (SQL Server PDW) topic.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_exec_connections (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the connections established to this instance of SQL Data Warehouse and the details of each connection.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**int**	Identifies the session associated with this connection. Use `SESSION_ID()` to return the `session_id` of the current connection.
connect_time	**datetime**	Timestamp when connection was established. Is not nullable.
encrypt_option	**nvarchar(40)**	Indicates TRUE (connection is encrypted) or FALSE (connection is not enctypred).
auth_scheme	**nvarchar(40)**	Specifies SQL Server/Windows Authentication scheme used with this connection. Is not nullable.
client_id	**varchar(48)**	IP address of the client connecting to this server. Is nullable.
sql_spid	**int**	The server process ID of the connection. Use `@@SPID` to return the `sql_spid` of the current connection.For most purposed, use the `session_id` instead.

Permissions

Requires **VIEW SERVER STATE** permission on the server.

Relationship Cardinalities

dm_pdw_exec_sessions.session_id	dm_pdw_exec_connections.session_id	One-to-one
dm_pdw_exec_requests.connection_id	dm_pdw_exec_connections.connection_id	Many to one

Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

Typical query to gather information about a queries own connection.

```
SELECT
    c.session_id, c.encrypt_option,
    c.auth_scheme, s.client_id, s.login_name,
    s.status, s.query_count
FROM sys.dm_pdw_exec_connections AS c
JOIN sys.dm_pdw_exec_sessions AS s
    ON c.session_id = s.session_id
WHERE c.session_id = SESSION_ID();
```

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_exec_requests (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all requests currently or recently active in SQL Data Warehouse. It lists one row per request/query.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
request_id	**nvarchar(32)**	Key for this view. Unique numeric id associated with the request.	Unique across all requests in the system.
session_id	**nvarchar(32)**	Unique numeric id associated with the session in which this query was run. See sys.dm_pdw_exec_sessions (Transact-SQL).	
status	**nvarchar(32)**	Current status of the request.	'Running', 'Suspended', 'Completed', 'Cancelled', 'Failed'.
submit_time	**datetime**	Time at which the request was submitted for execution.	Valid **datetime** smaller or equal to the current time and start_time.
start_time	**datetime**	Time at which the request execution was started.	NULL for queued requests; otherwise, valid **datetime** smaller or equal to current time.
end_compile_time	**datetime**	Time at which the engine completed compiling the request.	NULL for requests that have not been compiled yet; otherwise a valid **datetime** less than start_time and less than or equal to the current time.
end_time	**datetime**	Time at which the request execution completed, failed, or was cancelled.	Null for queued or active requests; otherwise, a valid **datetime** smaller or equal to current time.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
total_elapsed_time	**int**	Time elapsed in execution since the request was started, in milliseconds.	Between 0 and the difference between start_time and end_time. If total_elapsed_time exceeds the maximum value for an integer, total_elapsed_time will continue to be the maximum value. This condition will generate the warning "The maximum value has been exceeded." The maximum value in milliseconds is equivalent to 24.8 days.
label	**nvarchar(255)**	Optional label string associated with some SELECT query statements.	Any string containing 'a-z','A-Z','0-9','_'.
error_id	**nvarchar(36)**	Unique id of the error associated with the request, if any.	See sys.dm_pdw_errors (Transact-SQL); set to NULL if no error occurred.
database_id	**int**	Identifier of database used by explicit context (e.g., USE DB_X).	See id in sys.databases (Transact-SQL).
command	**nvarchar(4000)**	Holds the full text of the request as submitted by the user.	Any valid query or request text. Queries that are longer than 4000 bytes are truncated.
resource_class	**nvarchar(20)**	The resource class for this request. See related **concurrency_slots_used** in sys.dm_pdw_resource_waits (Transact-SQL).	SmallRC MediumRC LargeRC XLargeRC

For information about the maximum rows retained by this view, see "Minimum and Maximum Values" in the Parallel Data Warehouse product documentation.

Permissions

Requires VIEW SERVER STATE permission.

Security

sys.dm_pdw_exec_requests does not filter query results according to database-specific permissions. Logins with VIEW SERVER STATE permission can obtain results query results for all databases

> **WARNING**
>
> An attacker can use sys.dm_pdw_exec_requests to retrieve information about specific database objects by simply having VIEW SERVER STATE permission and by not having a database-specific permission.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_exec_sessions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all sessions currently or recently open on the appliance. It lists one row per session.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
session_id	**nvarchar(32)**	The id of the current query or the last query run (if the session is TERMINATED and the query was executing at time of termination). Key for this view.	Unique across all sessions in the system.
status	**nvarchar(10)**	For current sessions, identifies whether the session is currently active or idle. For past sessions, the session status may show closed or killed (if the session was forcibly closed).	'ACTIVE', 'CLOSED', 'IDLE', 'TERMINATED'
request_id	**nvarchar(32)**	The id of the current query or last query run.	Unique across all requests in the system. Null if none has been run.
security_id	**varbinary(85)**	Security ID of the principal running the session.	
login_name	**nvarchar(128)**	The login name of the principal running the session.	Any string conforming to the user naming conventions.
login_time	**datetime**	Date and time at which the user logged in and this session was created.	Valid **datetime** before current time.
query_count	**int**	Captures the number of queries/requeststhis session has run since creation.	Greater than or equal to 0.
is_transactional	**bit**	Captures whether a session is currently within a transaction or not.	0 for auto-commit, 1 for transactional.
client_id	**nvarchar(255)**	Captures client information for the session.	Any valid string.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
app_name	**nvarchar(255)**	Captures application name information optionally set as part of the connection process.	Any valid string.
sql_spid	**int**	The id number of the SPID. Use the `session_id` this session. Use the `sql_spid` column to join to **sys.dm_pdw_nodes_exec_ sessions**. ** **Warning** ** This column contains closed SPIDs.	

For information about the maximum rows retained by this view, see the Maximum System View Values section in the Minimum and Maximum Values (SQL Server PDW) topic.

Permissions

Requires the `VIEW SERVER STATE` permission.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_hadoop_operations (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Contains a row for each map-reduce job that is pushed down to Hadoop as part of running a SQL Data Warehouse query on an external Hadoop table. Each map-reduce job represents one of the predicates in the query. This is only used when predicate pushdown is enabled for queries on Hadoop external tables.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
request_id	**nvarchar(32)**	ID for this external Hadoop operation.	Same as ID in sys.dm_pdw_exec_requests (Transact-SQL).
step_index	**int**	Index of the query step that refers to this Hadoop operation.	Same as step_index in sys.dm_pdw_request_steps (Transact-SQL).
operation_type	**nvarchar(255)**	Describes the type of external operation.	'External Hadoop Operation'
operation_name	**nvarchar(4000)**	The job ID for a map-reduce job. This is returned by Hadoop after SQL Data Warehouse submits the job.	
map_progress	**float**	The percentage of input data that has been consumed so far by the map job.	A floating point number between, and including, 0 and 100.
reduce_progress	**int**	The percentage of the reduce job that has completed..	A floating point number between, and including, 0 and 100.

See Also

System Views (Transact-SQL)

sys.dm_pdw_lock_waits (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about the requests that are waiting for locks.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
wait_id	**bigint**	Position of the request in the waiting list.	0-based ordinal. This is not unique across all wait entries.
session_id	**nvarchar(32)**	ID of the session in which the wait state occurred.	See session_id in sys.dm_pdw_exec_sessions (Transact-SQL).
type	**nvarchar(255)**	Type of wait this entry represents.	Possible values: Shared SharedUpdate ExclusiveUpdate Exclusive
object_type	**nvarchar(255)**	Type of object that is affected by the wait.	Possible values: OBJECT DATABASE SYSTEM SCHEMA APPLICATION
object_name	**nvarchar(386)**	Name or GUID of the specified object that was affected by the wait.	Tables and views are displayed with three-part names. Indexes and statistics are displayed with four-part names. Names, principals, and databases are string names.
request_id	**nvarchar(32)**	ID of the request on which the wait state occurred.	ID of the request. This is a GUID for load requests.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
request_time	**datetime**	Time at which the lock or resource was requested.	
acquire_time	**datetime**	Time at which the lock or resource was acquired.	
state	**nvarchar(50)**	State of the wait state.	Information not available.
priority	**int**	Priority of the waiting item.	Information not available.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_network_credentials (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a list of all network credentials stored in the Parallel Data Warehouse appliance for all target servers. Results are listed for the Control node, and every Compute node.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	Unique numeric id associated with the node.
target_server_name	**nvarchar(32)**	IP address of the target server that Parallel Data Warehouse will access by using the username and password credentials.
username	**nvarchar(32)**	Username for which the password is stored.
last_modified	**datetime**	Datetime of the last operation that modified the credential.

Permissions

Requires VIEW SERVER STATE.

General Remarks

The key for this dynamic management view is *pdw_node_id* plus *target_server_name*.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_node_status (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds additional information (over sys.dm_pdw_nodes (Transact-SQL)) about the performance and status of all appliance nodes. It lists one row per node in the appliance.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
pdw_node_id	**int**	Unique numeric id associated with the node. Key for this view.	Unique across the appliance, regardless of type.
process_id	**int**	Information not available.	
process_name	**nvarchar(255)**	Information not available.	
allocated_memory	**bigint**	Total allocated memory on this node.	
available_memory	**bigint**	Total available memory on this node.	
process_cpu_usage	**bigint**	Total process CPU usage, in ticks.	
total_cpu_usage	**bigint**	Total CPU usage, in ticks.	
thread_count	**bigint**	Total number of threads in use on this node.	
handle_count	**bigint**	Total number of handles in use on this node.	
total_elapsed_time	**bigint**	Total time elapsed since system start or restart.	Total time elapsed since system start or restart. If total_elapsed_time exceeds the maximum value for an integer (24.8 days in milliseconds), it will cause materialization failure due to overflow. The maximum value in milliseconds is equivalent to 24.8 days.
is_available	**bit**	Flag indicating whether this node is available.	

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
sent_time	**datetime**	Last time a network package was sent by this node.	
received_time	**datetime**	Last time a network package was received by this node.	
error_id	**nvarchar(36)**	Unique identifier of the last error that occurred on this node.	

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_nodes (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all of the nodes in Analytics Platform System. It lists one row per node in the appliance.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
pdw_node_id	**int**	Unique numeric id associated with the node. Key for this view.	Unique across the appliance, regardless of type.
type	**nvarchar(32)**	Type of the node.	'COMPUTE', 'CONTROL', 'MANAGEMENT'
name	**nvarchar(32)**	Logical name of the node.	Any string of appropriate length.
address	**nvarchar(32)**	IP address of this node.	In the format of [0-255].[0-255].[0-255].[0-255].
is_passive	**int**	Indicates whether the virtual machine running the node is running on the assigned server or has failed over to the spare server.	0 – node VM is running on the original server. 1 – node VM is running on the spare server.
region	**nvarchar(32)**	The region where the node is running.	'PDW', 'HDINSIGHT'

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_nodes_database_encryption_keys (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the encryption state of a database and its associated database encryption keys. **sys.dm_pdw_nodes_database_encryption_keys** provides this information for each node. For more information about database encryption, see Transparent Data Encryption (SQL Server PDW).

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	ID of the physical database on each node.
encryption_state	**int**	Indicates whether the database on this node is encrypted or not encrypted. 0 = No database encryption key present, no encryption 1 = Unencrypted 2 = Encryption in progress 3 = Encrypted 4 = Key change in progress 5 = Decryption in progress 6 = Protection change in progress (The certificate that is encrypting the database encryption key is being changed.)
create_date	**datetime**	Displays the date the encryption key was created.
regenerate_date	**datetime**	Displays the date the encryption key was regenerated.
modify_date	**datetime**	Displays the date the encryption key was modified.
set_date	**datetime**	Displays the date the encryption key was applied to the database.
opened_date	**datetime**	Shows when the database key was last opened.
key_algorithm	**varchar(?)**	Displays the algorithm that is used for the key.

COLUMN NAME	DATA TYPE	DESCRIPTION
key_length	**int**	Displays the length of the key.
encryptor_thumbprint	**varbin**	Shows the thumbprint of the encryptor.
percent_complete	**real**	Percent complete of the database encryption state change. This will be 0 if there is no state change.
node_id	**int**	Unique numeric id associated with the node.

Permissions

Requires the VIEW SERVER STATE permission on the server.

Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example joins `sys.dm_pdw_nodes_database_encryption_keys` to other system tables to indicate the encryption state for each node of the TDE protected databases.

```
SELECT D.database_id AS DBIDinMaster, D.name AS UserDatabaseName,
PD.pdw_node_id AS NodeID, DM.physical_name AS PhysDBName,
keys.encryption_state
FROM sys.dm_pdw_nodes_database_encryption_keys AS keys
JOIN sys.pdw_nodes_pdw_physical_databases AS PD
    ON keys.database_id = PD.database_id AND keys.pdw_node_id = PD.pdw_node_id
JOIN sys.pdw_database_mappings AS DM
    ON DM.physical_name = PD.physical_name
JOIN sys.databases AS D
    ON D.database_id = DM.database_id
ORDER BY D.database_id, PD.pdw_node_ID;
```

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)
CREATE DATABASE ENCRYPTION KEY (Transact-SQL)
ALTER DATABASE ENCRYPTION KEY (Transact-SQL)
DROP DATABASE ENCRYPTION KEY (Transact-SQL)

sys.dm_pdw_os_event_logs (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information regarding the different Windows Event logs on the different nodes.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
pdw_node_id	**int**	Appliance node this log is from. pdw_node_id and log_name form the key for this view.	
log_name	**nvarchar(255)**	Windows event log name. pdw_node_id and log_name form the key for this view.	
log_source	**nvarchar(255)**	Windows event log source name.	
event_id	**int**	ID of the event. Not unique.	
event_type	**nvarchar(255)**	Type of the event, identifying severity.	'Information', 'Warning', 'Error'
event_message	**nvarchar(4000)**	Details of the event.	
generate_time	**datetime**	Time the event was created.	
write_time	**datetime**	Time the event was actually written to the log.	

For information about the maximum rows retained by this view, see the Maximum System View Values section in the Minimum and Maximum Values (SQL Server PDW) topic.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_os_performance_counters (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Contains information about Windows performance counters for the nodes in Parallel Data Warehouse.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
pdw_node_id	**int**	The ID of the node that contains the counter. pdw_node_id and counter_name form the key for this view.	See node_id in sys.dm_pdw_nodes (Transact-SQL).
counter_name	**nvarchar(255)**	Name of Windows performance counter.	
counter_category	**nvarchar(255)**	Name of Windows performance counter category.	
instance_name	**nvarchar(255)**	Name of the specific instance of the counter.	
counter_value	**Decimal(38,10)**	Current value of the counter.	
last_update_time	**Datetime2(3)**	Timestamp of last time the value was updated.	

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_os_threads (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
pdw_node_id	**int**	The ID of the affected node. pdw_node_id and thread_id form the key for this view.	See node_id in sys.dm_pdw_nodes (Transact-SQL).
thread_id	**int**	pdw_node_id and thread_id form the key for this view.	
process_id	**int**		
name	**nvarchar(255)**		
priority	**int**		
start_time	**datetime**		
state	**nvarchar(32)**		
wait_reason	**nvarchar(32)**		
total_processor_elapsed_time	**bigint**	Total kernel time used by the thread.	
total_user_elapsed_time	**bigint**	Total user time used by the thread	

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_query_stats_xe (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This DMV is deprecated and will be removed in a future release. In this release, it returns 0 rows.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
event	**nvarchar(60)**	Key for this view.	
event_id	**nvarchar(36)**		
create_time	**datetime**		
session_id	**int**	The id for the session.	See session_id in sys.dm_pdw_exec_sessions (Transact-SQL).
cpu	**int**		
reads	**int**	Number of logical reads since the start of the event.	
writes	**int**	Number of logical writes since the start of the event.	
sql_text	**nvarchar(4000)**		
client_app_name	**nvarchar(255)**		
tsql_stack	**nvarchar(255)**		
pdw_node_id	**int**	Node on which this Xevent instance is running.	

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_query_stats_xe_file (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This DMV is deprecated and will be removed in a future release. In this release, it returns 0 rows.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
event	**nvarchar(60)**	Key for this view.	
data	**xml**		
pdw_node_id	**int**	Node on which this Xevent instance is running.	

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_request_steps (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all steps that compose a given request or query in SQL Data Warehouse. It lists one row per query step.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
request_id	**nvarchar(32)**	request_id and step_index make up the key for this view. Unique numeric id associated with the request.	See request_id in sys.dm_pdw_exec_requests (Transact-SQL).
step_index	**int**	request_id and step_index make up the key for this view. The position of this step in the sequence of steps that make up the request.	0 to (n-1) for a request with n steps.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
operation_type	**nvarchar(35)**	Type of operation represented by this step.	**DMS query plan operations:** 'ReturnOperation', 'PartitionMoveOperation', 'MoveOperation', 'BroadcastMoveOperation', 'ShuffleMoveOperation', 'TrimMoveOperation', 'CopyOperation', 'DistributeReplicatedTableMoveOperation' **SQL query plan operations:** 'OnOperation', 'RemoteOperation' **Other query plan operations:** 'MetaDataCreateOperation', 'RandomIDOperation' **External operations for reads:** 'HadoopShuffleOperation', 'HadoopRoundRobinOperation', 'HadoopBroadcastOperation' **External operations for MapReduce:** 'HadoopJobOperation', 'HdfsDeleteOperation' **External operations for writes:** 'ExternalExportDistributedOperation', 'ExternalExportReplicatedOperation', 'ExternalExportControlOperation' For more information, see "Understanding Query Plans" in the Parallel Data Warehouse product documentation.
distribution_type	**nvarchar(32)**	Type of distribution this step will undergo.	'AllNodes', 'AllDistributions', 'AllComputeNodes', 'ComputeNode', 'Distribution', 'SubsetNodes', 'SubsetDistributions', 'Unspecified'
location_type	**nvarchar(32)**	Where the step is running.	'Compute', 'Control', 'DMS'

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
status	**nvarchar(32)**	Status of this step.	Pending, Running, Complete, Failed, UndoFailed, PendingCancel, Cancelled, Undone, Aborted
error_id	**nvarchar(36)**	Unique id of the error associated with this step, if any.	See error_id of sys.dm_pdw_errors (Transact-SQL). NULL if no error occurred.
start_time	**datetime**	Time at which the step started execution.	Smaller or equal to current time and larger or equal to end_compile_time of the query to which this step belongs. For more information on queries, see sys.dm_pdw_exec_requests (Transact-SQL).
end_time	**datetime**	Time at which this step completed execution, was cancelled, or failed.	Smaller or equal to current time and larger or equal to start_time. Set to NULL for steps currently in execution or queued.
total_elapsed_time	**int**	Total amount of time the query step has been running, in milliseconds.	Between 0 and the difference between end_time and start_time. 0 for queued steps. If total_elapsed_time exceeds the maximum value for an integer, total_elapsed_time will continue to be the maximum value. This condition will generate the warning "The maximum value has been exceeded." The maximum value in milliseconds is equivalent to 24.8 days.
row_count	**bigint**	Total number of rows changed or returned by this request.	0 for steps that did not change or return data. Otherwise, number of rows affected.
command	**nvarchar(4000)**	Holds the full text of the command of this step.	Any valid request string for a step. NULL when the operation is of the type MetaDataCreateOperation. Truncated if longer than 4000 characters.

For information about the maximum rows retained by this view, see the Maximum System View Values section in the "Minimum and Maximum Values" in the Parallel Data Warehouse product documentation.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_resource_waits (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Displays wait information for all resource types in SQL Data Warehouse.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
wait_id	**bigint**	Position of the request in the waiting list.	0-based ordinal. This is not unique across all wait entries.
session_id	**nvarchar(32)**	ID of the session in which the wait state occurred.	See session_id in sys.dm_pdw_exec_sessions (Transact-SQL).
type	**nvarchar(255)**	Type of wait this entry represents.	Possible values: Connection Local Queries Concurrency Distributed Queries Concurrency DMS Concurrency Backup Concurrency
object_type	**nvarchar(255)**	Type of object that is affected by the wait.	Possible values: **OBJECT** **DATABASE** **SYSTEM** **SCHEMA** **APPLICATION**
object_name	**nvarchar(386)**	Name or GUID of the specified object that was affected by the wait.	Tables and views are displayed with three-part names. Indexes and statistics are displayed with four-part names. Names, principals, and databases are string names.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
request_id	**nvarchar(32)**	ID of the request on which the wait state occurred.	QID identifier of the request. GUID identifier for load requests.
request_time	**datetime**	Time at which the lock or resource was requested.	
acquire_time	**datetime**	Time at which the lock or resource was acquired.	
state	**nvarchar(50)**	State of the wait state.	Information not available.
priority	**int**	Priority of the waiting item.	Information not available.
concurrency_slots_used	**int**	Number of concurrency slots (32 max) reserved for this request.	1 – for SmallRC 3 – for MediumRC 7 for LargeRC 22 – for XLargeRC
resource_class	**nvarchar(20)**	The resource class for this request.	SmallRC MediumRC LargeRC XLargeRC

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_sql_requests (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all SQL Server query distributions as part of a SQL step in the query.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
request_id	**nvarchar(32)**	Unique identifier of the query to which this SQL query distribution belongs. request_id, step_index, and distribution_id form the key for this view.	See request_id in sys.dm_pdw_exec_requests (Transact-SQL).
step_index	**int**	Index of the query step this distribution is part of. request_id, step_index, and distribution_id form the key for this view.	See step_index in sys.dm_pdw_request_steps (Transact-SQL).
pdw_node_id	**int**	Unique identifier of the node on which this query distribution is run.	See node_id in sys.dm_pdw_nodes (Transact-SQL).
distribution_id	**int**	Unique identifier of the distribution on which this query distribution is run. request_id, step_index, and distribution_id form the key for this view.	See distribution_id in sys.pdw_distributions (Transact-SQL). Set to -1 for requests that run at the node scope, not the distribution scope.
status	**nvarchar(32)**	Current status of the query distribution.	Pending, Running, Failed, Cancelled, Complete, Aborted, CancelSubmitted
error_id	**nvarchar(36)**	Unique identifier of the error associated with this query distribution, if any.	See error_id in sys.dm_pdw_errors (Transact-SQL). Set to NULL if no error occurred.
start_time	**datetime**	Time at which query distribution started execution.	Smaller or equal to current time and greater or equal to start_time of the query step this query distribution belongs to

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
end_time	**datetime**	Time at which this query distribution completed execution, was cancelled, or failed.	Greater or equal to start time, or set to NULL if the query distribution is ongoing or queued.
total_elapsed_time	**int**	Represents the time the query distribution has been running, in milliseconds.	Greater or equal to 0. Equal to the delta of start_time and end_time for completed, failed, or cancelled query distributions. If total_elapsed_time exceeds the maximum value for an integer, total_elapsed_time will continue to be the maximum value. This condition will generate the warning "The maximum value has been exceeded." The maximum value in milliseconds is equivalent to 24.8 days.
row_count	**bigint**	Number of rows changed or read by this query distribution.	-1 for operations that do not change or return data, such as CREATE TABLE and DROP TABLE.
spid	**int**	Session id on the SQL Server instance running the query distribution.	
command	**nvarchar(4000)**	Full text of command for this query distribution.	Any valid query or request string.

For information about the maximum rows retained by this view, see the Maximum System View Values section in the Minimum and Maximum Values (SQL Server PDW) topic.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_sys_info (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Provides a set of appliance-level counters that reflect overall activity on the appliance.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
total_sessions	**int**	Number of sessions currently in the system.	0 to max_active_sessions (see below).
idle_sessions	**int**	Number of sessions currently idle.	
active_requests	**int**	Number of active requests currently running.	
queued_requests	**int**	Number of currently queued requests.	
active_loads	**int**	Number of loads currently running in the system.	
queued_loads	**int**	Number of queued loads waiting for execution.	
active_backups	**int**	Number of backups currently running.	
active_restores	**int**	Number of backup restores currently running.	

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)

sys.dm_pdw_wait_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information related to the SQL Server OS state related to instances running on the different nodes. For a list of waits types and their description, see sys.dm_os_wait_stats.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
pdw_node_id	**int**	ID of the node this entry refers to.	
wait_name	**nvarchar(255)**	Name of the wait type.	
max_wait_time	**bigint**	Maximum wait time of this wait type.	
request_count	**bigint**	Number of waits of this wait type outstanding.	
signal_time	**bigint**	Difference between the time that the waiting thread was signaled and when it started running.	
completed_count	**bigint**	Total number of waits of this type completed since the last server restart.	
wait_time	**bigint**	Total wait time for this wait type in millisecons. Inclusive of signal_time.	

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)
sys.dm_pdw_waits (Transact-SQL)

sys.dm_pdw_waits (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Holds information about all wait states encountered during execution of a request or query, including locks, waits on transmission queues, and so on.

COLUMN NAME	DATA TYPE	DESCRIPTION	RANGE
wait_id	**bigint**	Unique numeric id associated with the wait state. Key for this view.	Unique across all waits in the system.
session_id	**nvarchar(32)**	ID of the session on which the wait state occurred.	See session_id in sys.dm_pdw_exec_sessions (Transact-SQL).
type	**nvarchar(255)**	Type of wait this entry represents.	Information not available.
object_type	**nvarchar(255)**	Type of object that is affected by the wait.	Information not available.
object_name	**nvarchar(386)**	Name or GUID of the specified object that was affected by the wait.	
request_id	**nvarchar(32)**	ID of the request on which the wait state occurred.	See request_id in sys.dm_pdw_exec_requests (Transact-SQL).
request_time	**datetime**	Time at which the wait state was requested.	
acquire_time	**datetime**	Time at which the lock or resource was acquired.	
state	**nvarchar(50)**	State of the wait state.	Information not available.
priority	**int**	Priority of the waiting item.	Information not available.

See Also

SQL Data Warehouse and Parallel Data Warehouse Dynamic Management Views (Transact-SQL)
sys.dm_pdw_wait_stats (Transact-SQL)

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the dynamic management views that are associated with the SQL Server Operating System (SQLOS). The SQLOS is responsible for managing operating system resources that are specific to SQL Server.

sys.dm_os_buffer_descriptors (Transact-SQL)	sys.dm_os_memory_pools (Transact-SQL)
sys.dm_os_child_instances (Transact-SQL)	sys.dm_os_nodes (Transact-SQL)
sys.dm_os_cluster_nodes (Transact-SQL)	sys.dm_os_performance_counters (Transact-SQL)
sys.dm_os_dispatcher_pools (Transact-SQL)	sys.dm_os_process_memory (Transact-SQL)
sys.dm_os_host_info (Transact-SQL)	sys.dm_os_schedulers (Transact-SQL)
sys.dm_os_hosts (Transact-SQL)	sys.dm_os_stacks (Transact-SQL)
sys.dm-os-job-object (Transact-SQL)	
sys.dm_os_latch_stats (Transact-SQL)	sys.dm_os_sys_info (Transact-SQL)
sys.dm_os_loaded_modules (Transact-SQL)	sys.dm_os_sys_memory (Transact-SQL)
sys.dm_os_memory_brokers (Transact-SQL)	sys.dm_os_tasks (Transact-SQL)
sys.dm_os_memory_cache_clock_hands (Transact-SQL)	sys.dm_os_threads (Transact-SQL)
sys.dm_os_memory_cache_counters (Transact-SQL)	sys.dm_os_virtual_address_dump (Transact-SQL)
sys.dm_os_memory_cache_entries (Transact-SQL)	sys.dm_os_volume_stats (Transact-SQL)
sys.dm_os_memory_cache_hash_tables (Transact-SQL)	sys.dm_os_wait_stats (Transact-SQL)
sys.dm_os_memory_clerks (Transact-SQL)	sys.dm_os_waiting_tasks (Transact-SQL)
sys.dm_os_memory_nodes (Transact-SQL)	sys.dm_os_windows_info (Transact-SQL)
sys.dm_os_memory_objects (Transact-SQL)	sys.dm_os_workers (Transact-SQL)

The following SQL Server Operating System–related dynamic management views are Identified for informational purposes only. Not supported. Future compatibility is not guaranteed..

sys.dm_os_function_symbolic_name	**sys.dm_os_ring_buffers**
sys.dm_os_memory_allocations	**sys.dm_os_sublatches**
sys.dm_os_worker_local_storage	

See Also

Dynamic Management Views and Functions (Transact-SQL)

sys.dm_os_buffer_descriptors (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about all the data pages that are currently in the SQL Server buffer pool. The output of this view can be used to determine the distribution of database pages in the buffer pool according to database, object, or type. In SQL Server 2017, this dynamic management view also returns information about the data pages in the buffer pool extension file. For more information, see Buffer Pool Extension.

When a data page is read from disk, the page is copied into the SQL Server buffer pool and cached for reuse. Each cached data page has one buffer descriptor. Buffer descriptors uniquely identify each data page that is currently cached in an instance of SQL Server. sys.dm_os_buffer_descriptors returns cached pages for all user and system databases. This includes pages that are associated with the Resource database.

> **NOTE:** To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_buffer_descriptors**.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	ID of database associated with the page in the buffer pool. Is nullable.
file_id	**int**	ID of the file that stores the persisted image of the page. Is nullable.
page_id	**int**	ID of the page within the file. Is nullable.
page_level	**int**	Index level of the page. Is nullable.
allocation_unit_id	**bigint**	ID of the allocation unit of the page. This value can be used to join sys.allocation_units. Is nullable.
page_type	**nvarchar(60)**	Type of the page, such as: Data page or Index page. Is nullable.
row_count	**int**	Number of rows on the page. Is nullable.
free_space_in_bytes	**int**	Amount of available free space, in bytes, on the page. Is nullable.
is_modified	**bit**	1 = Page has been modified after it was read from the disk. Is nullable.
numa_node	**int**	Nonuniform Memory Access node for the buffer. Is nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
read_microsec	**bigint**	The actual time (in microseconds) required to read the page into the buffer. This number is reset when the buffer is reused. Is nullable.
is_in_bpool_extension	**bit**	1 = Page is in buffer pool extension. Is nullable.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

sys.dm_os_buffer_descriptors returns pages that are being used by the Resource database.
sys.dm_os_buffer_descriptors does not return information about free or stolen pages, or about pages that had errors when they were read.

FROM	TO	ON	RELATIONSHIP
sys.dm_os_buffer_descriptors	sys.databases	database_id	many-to-one
sys.dm_os_buffer_descriptors	<userdb>.sys.allocation_units	allocation_unit_id	many-to-one
sys.dm_os_buffer_descriptors	<userdb>.sys.database_files	file_id	many-to-one
sys.dm_os_buffer_descriptors	sys.dm_os_buffer_pool_extension_configuration	file_id	many-to-one

Examples

A. Returning cached page count for each database

The following example returns the count of pages loaded for each database.

```
SELECT COUNT(*)AS cached_pages_count
    ,CASE database_id
        WHEN 32767 THEN 'ResourceDb'
        ELSE db_name(database_id)
        END AS database_name
FROM sys.dm_os_buffer_descriptors
GROUP BY DB_NAME(database_id) ,database_id
ORDER BY cached_pages_count DESC;
```

B. Returning cached page count for each object in the current database

The following example returns the count of pages loaded for each object in the current database.

```
SELECT COUNT(*)AS cached_pages_count
    ,name ,index_id
FROM sys.dm_os_buffer_descriptors AS bd
    INNER JOIN
    (
        SELECT object_name(object_id) AS name
            ,index_id ,allocation_unit_id
        FROM sys.allocation_units AS au
            INNER JOIN sys.partitions AS p
                ON au.container_id = p.hobt_id
                    AND (au.type = 1 OR au.type = 3)
        UNION ALL
        SELECT object_name(object_id) AS name
            ,index_id, allocation_unit_id
        FROM sys.allocation_units AS au
            INNER JOIN sys.partitions AS p
                ON au.container_id = p.partition_id
                    AND au.type = 2
    ) AS obj
        ON bd.allocation_unit_id = obj.allocation_unit_id
WHERE database_id = DB_ID()
GROUP BY name, index_id
ORDER BY cached_pages_count DESC;
```

See Also

sys.allocation_units (Transact-SQL)

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)
Resource Database
sys.dm_os_buffer_pool_extension_configuration (Transact-SQL)

sys.dm_os_buffer_pool_extension_configuration (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2014) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns configuration information about the buffer pool extension in SQL Server. Returns one row for each buffer pool extension file.

COLUMN NAME	DATA TYPE	DESCRIPTION
path	**nvarchar**(256)	Path and file name of the buffer pool extension cache. Nullable.
file_id	**int**	ID of the buffer pool extension file. Is not nullable.
state	**int**	The state of the buffer pool extension feature. Is not nullable. 0 - Buffer pool extension disabled 1 - Buffer pool extension disabling 2 - Reserved for the future use 3 - Buffer pool extension enabling 4 - Reserved for the future use 5 - Buffer pool extension enabled
state_description	**nvarchar**(60)	Describes the state of the buffer pool extension feature. Is nullable. 0 = BUFFER POOL EXTENSION DISABLED 1 = BUFFER POOL EXTENSION ENABLED
current_size_in_kb	**bigint**	Current size of the buffer pool extension file. Is not nullable.

Permissions

Requires VIEW SERVER STATE permission on the server.

Examples

A. Returning configuration buffer pool extension information

The following example returns all columns from the sys.dm_os_buffer_pool_extension_configuration DMV.

```sql
SELECT path, file_id, state, state_description, current_size_in_kb
FROM sys.dm_os_buffer_pool_extension_configuration;
```

B. Returning the number of cached pages in the buffer pool extension file

The following example returns the number of cached pages in each buffer pool extension file.

```sql
SELECT COUNT(*) AS cached_pages_count
FROM sys.dm_os_buffer_descriptors
WHERE is_in_bpool_extension <> 0
;
```

See Also

Buffer Pool Extension
sys.dm_os_buffer_descriptors (Transact-SQL)

sys.dm_os_child_instances (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each user instance that has been created from the parent server instance.

> **IMPORTANT!** This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

The information returned from **sys.dm_os_child_instances** can be used to determine the state of each User Instance (heart_beat) and to obtain the pipe name (instance_pipe_name) that can be used to create a connection to the User Instance using SQL Server Management Studio or SQLCmd. You can only connect to a User Instance after it has been started by an external process, such as a client application. SQL management tools cannot start a User Instance.

> **NOTE:** User Instances are a feature of SQL Server 2012 Express only.
>
> **NOTE** To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_child_instances**.

COLUMN	DATA TYPE	DESCRIPTION
owning_principal_name	**nvarchar(256)**	The name of the user that this user instance was created for.
owning_principal_sid	nvarchar(256)	SID (Security-Identifier) of the principal who owns this user instance. This matches Windows SID.
owning_principal_sid_binary	varbinary(85)	Binary version of the SID for the user who owns the user Instance
instance_name	**nvarchar(128)**	The name of this user instance.
instance_pipe_name	**nvarchar(260)**	When a user instance is created, a named pipe is created for applications to connect to. This name can be used in a connect string to connect to this user instance.
os_process_id	**Int**	The process number of the Windows process for this user instance.
os_process_creation_date	**Datetime**	The date and time when this user instance process was last started.
heart_beat	**nvarchar(5)**	Current state of this user instance; either ALIVE or DEAD.

COLUMN	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

Requires VIEW SERVER STATE permission on the server.

Remarks

For more information about dynamic management view, see Dynamic Management Views and Functions (Transact-SQL) in SQL Server Books Online.

See Also

User Instances for Non-Administrators

sys.dm_os_cluster_nodes (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row for each node in the failover cluster instance configuration. If the current instance is a failover clustered instance, it returns a list of nodes on which this failover cluster instance (formerly "virtual server") has been defined. If the current server instance is not a failover clustered instance, it returns an empty rowset.

> **NOTE:** To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_cluster_nodes**.

COLUMN NAME	DATA TYPE	DESCRIPTION
NodeName	**sysname**	Name of a node in the SQL Server failover cluster instance (virtual server) configuration.
status	**int**	Status of the node in a SQL Server failover cluster instance: 0, 1, 2, 3, -1. For more information, see GetClusterNodeState Function.
status_description	**nvarchar(20)**	Description of the status of the SQL Server failover cluster node. 0 = up 1 = down 2 = paused 3 = joining -1 = unknown
is_current_owner	bit	1 means this node is the current owner of the SQL Server failover cluster resource.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Remarks

When failover clustering is enabled, the SQL Server instance can run on any of the nodes of the failover cluster that are designated as part of the SQL Server failover cluster instance (virtual server) configuration.

> **NOTE:** This view replaces the fn_virtualservernodes function, which will be deprecated in a future release.

Permissions

Requires VIEW SERVER STATE permission on the instance of SQL Server.

Examples

The following example uses sys. dm_os_cluster_nodes to return the nodes on a clustered server instance.

```
SELECT NodeName, status, status_description, is_current_owner
FROM sys.dm_os_cluster_nodes;
```

Here is the result set.

NODENAME	STATUS	STATUS_DESCRIPTION	IS_CURRENT_OWNER
node1	0	up	1
node2	0	up	0
Node3	1	down	0

See Also

sys.dm_os_cluster_properties (Transact-SQL)
sys.dm_io_cluster_shared_drives (Transact-SQL)
sys.fn_virtualservernodes (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)

sys.dm_os_cluster_properties (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row with the current settings for the SQL Server cluster resource properties identified in this topic. No data is returned if this view is run on a stand-alone instance of SQL Server.

These properties are used to set the values that affect failure detection, failure response time, and the logging for monitoring the health status of the SQL Server failover cluster instance.

COLUMN NAME	PROPERTY	DESCRIPTION
VerboseLogging	bigint	The logging level for the SQL Server failover cluster. Verbose logging can be turned on to provide additional details in the error logs for troubleshooting. One of the following values: 0 – Logging is turned off (default) 1 - Errors only 2 – Errors and warnings For more information, see ALTER SERVER CONFIGURATION (Transact-SQL).
SqlDumperDumpFlags	bigint	SQLDumper dump flags determine the type of dump files generated by SQL Server. The default setting is 0.
SqlDumperDumpPath	nvarchar(260)	The location where the SQLDumper utility generates the dump files.
SqlDumperDumpTimeOut	bigint	The time-out value in milliseconds for the SQLDumper utility to generate a dump in case of a SQL Server failure. The default value is 0.
FailureConditionLevel	bigint	Sets the conditions under which the SQL Server failover cluster should fail or restart. The default value is 3. For a detailed explanation or to change the property settings, see Configure FailureConditionLevel Property Settings.

COLUMN NAME	PROPERTY	DESCRIPTION
HealthCheckTimeout	bigint	The time-out value for how long the SQL Server Database Engine resource DLL should wait for the server health information before it considers the instance of SQL Server as unresponsive. The time-out value is expressed in milliseconds. Default is 60000. For more information or to change this property setting, see Configure HealthCheckTimeout Property Settings.

Permissions

Requires VIEW SERVER STATE permissions on the SQL Server failover cluster instance.

Examples

The following example uses sys.dm_os_cluster_properties to return the property settings for the SQL Server failover cluster resource.

```
SELECT VerboseLogging, SqlDumperDumpFlags, SqlDumperDumpPath, SqlDumperDumpTimeOut, FailureConditionLevel,
HealthCheckTimeout
FROM sys.dm_os_cluster_properties;
```

Here is a sample result set.

VERBOSELOGGING	SQLDUMPERDUMPFLAGS	SQLDUMPERDUMPPATH	SQLDUMPERDUMPTIMEOUT	FAILURECONDITIONLEVEL	HEALTHCHECKTIMEOUT
0	0	NULL	0	3	60000

sys.dm_os_dispatcher_pools (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about session dispatcher pools. Dispatcher pools are thread pools used by system components to perform background processing.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_dispatcher_pools**.

COLUMN NAME	DATA TYPE	DESCRIPTION
dispatcher_pool_address	**varbinary(8)**	The address of the dispatcher pool. dispatcher_pool_address is unique. Is not nullable.
type	**nvarchar(256)**	The type of the dispatcher pool. Is not nullable. There are two types of dispatcher pools: DISP_POOL_XE_ENGINE DISP_POOL_XE_SESSION Query the DMV for the full list
name	**nvarchar(256)**	The name of the dispatcher pool. Is not nullable.
dispatcher_count	**int**	The number of active dispatcher threads. Is not nullable.
dispatcher_ideal_count	**int**	The number of dispatcher threads that the dispatcher pool can grow to use. Is not nullable.
dispatcher_timeout_ms	**int**	The time, in milliseconds, that a dispatcher will wait for new work before exiting. Is not nullable.
dispatcher_waiting_count	**int**	The number of idle dispatcher threads. Is not nullable.
queue_length	**int**	The number of work items waiting to be handled by the dispatcher pool. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

sys.dm_os_host_info (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2017) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row that displays operating system version information.

COLUMN NAME	DATA TYPE	DESCRIPTION
host_platform	**nvarchar(256)**	The type of operating system: Windows or Linux
host_distribution	**nvarchar(256)**	Description of the operating system.
host_release	**nvarchar(256)**	Microsoft Windows operating system release (version number). For a list of values and descriptions, see Operating System Version (Windows). For Linux, returns an empty string.
host_service_pack_level	**nvarchar(256)**	Service pack level of the Windows operating system. For Linux, returns an empty string.
host_sku	**int**	Windows Stock Keeping Unit (SKU) ID. For a list of SKU IDs and descriptions, see GetProductInfo Function. Is nullable. For Linux, returns NULL.
os_language_version	**int**	Windows locale identifier (LCID) of the operating system. For a list of LCID values and descriptions, see Locale IDs Assigned by Microsoft. Cannot be null.

Remarks

This view is similar to sys.dm_os_windows_info, adding columns to differentiate Windows and Linux.

Security

Permissions

The `SELECT` permission on `sys.dm_os_host_info` is granted to the `public` role by default. If revoked, requires `VIEW SERVER STATE` permission on the server.

Caution

Beginning with version SQL Server 2017 (14.x) CTP 1.3, SQL Server Management Studio version 17 requires `SELECT` permission on `sys.dm_os_host_info` in order to connect to SQL Server. If `SELECT` permission is revoked from `public`, only logins with `VIEW SERVER STATE` permission can connect with the newest version of SSMS. (Other tools, such as `sqlcmd.exe` can connect without `SELECT` permission on `sys.dm_os_host_info`.)

Examples

The following example returns all columns from the **sys.dm_os_host_info** view.

```
SELECT host_platform, host_distribution, host_release,
    host_service_pack_level, host_sku, os_language_version
FROM sys.dm_os_host_info;
```

Here is a sample result set on Windows:

HOST_PLATFORM	HOST_DISTRIBUTION	HOST_RELEASE	HOST_SERVICE_PACK_LEVEL	HOST_SKU	OS_LANGUAGE_VERSION
Windows	Windows Server 2012 R2 Standard	6.3		7	1033

Here is a sample result set on Linux:

HOST_PLATFORM	HOST_DISTRIBUTION	HOST_RELEASE	HOST_SERVICE_PACK_LEVEL	HOST_SKU	OS_LANGUAGE_VERSION
Linux	Ubuntu	16.04		NULL	1033

See Also

sys.dm_os_sys_info (Transact-SQL)
sys.dm_os_windows_info (Transact-SQL)

sys.dm_os_hosts (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns all the hosts currently registered in an instance of SQL Server. This view also returns the resources that are used by these hosts.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_hosts**.

COLUMN NAME	DATA TYPE	DESCRIPTION
host_address	**varbinary(8)**	Internal memory address of the host object.
type	**nvarchar(60)**	Type of hosted component. For example, SOSHOST_CLIENTID_SERVERSNI= SQL Server Native Interface SOSHOST_CLIENTID_SQLOLEDB = SQL Server Native Client OLE DB Provider SOSHOST_CLIENTID_MSDART = Microsoft Data Access Run Time
name	**nvarchar(32)**	Name of the host.
enqueued_tasks_count	**int**	Total number of tasks that this host has placed onto queues in SQL Server.
active_tasks_count	**int**	Number of currently running tasks that this host has placed onto queues.
completed_ios_count	**int**	Total number of I/Os issued and completed through this host.
completed_ios_in_bytes	**bigint**	Total byte count of the I/Os completed through this host.
active_ios_count	**int**	Total number of I/O requests related to this host that are currently waiting to complete.
default_memory_clerk_address	**varbinary(8)**	Memory address of the memory clerk object associated with this host. For more information, see sys.dm_os_memory_clerks (Transact-SQL).

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

SQL Server allows components, such as an OLE DB provider, that are not part of the SQL Server executable to allocate memory and participate in non-preemptive scheduling. These components are hosted by SQL Server, and all resources allocated by these components are tracked. Hosting allows SQL Server to better account for resources used by components external to the SQL Server executable.

Relationship Cardinalities

FROM	TO	RELATIONSHIP
sys.dm_os_hosts. default_memory_clerk_address	sys.dm_os_memory_clerks. memory_clerk_address	one to one
sys.dm_os_hosts. host_address	sys.dm_os_memory_clerks. host_address	one to one

Examples

The following example determines the total amount of memory committed by a hosted component.

Applies to: SQL Server 2012 (11.x) through SQL Server 2017.

```
SELECT h.type, SUM(mc.pages_kb) AS commited_memory
FROM sys.dm_os_memory_clerks AS mc
INNER JOIN sys.dm_os_hosts AS h
    ON mc.memory_clerk_address = h.default_memory_clerk_address
GROUP BY h.type;
```

See Also

sys.dm_os_memory_clerks (Transact-SQL)
SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_job_object (Azure SQL Database)

5/3/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a single row describing the configuration of the job object that manages the SQL Server process, as well as certain resource consumption statistics at the job object level. Returns an empty set if SQL Server is not running in a job object.

A job object is a Windows construct that implements CPU, memory, and IO resource governance at the operating system level. For more information about job objects, see Job Objects.

> **NOTE**
>
> The sys.dm_os_job_object DMV may currently appear as sys.dm_job_object. This is temporary: `sys.dm_os_job_object` will be the permanent name of this DMV.

COLUMNS	DATA TYPE	DESCRIPTION
cpu_rate	**int**	Specifies the portion of processor cycles that the SQL Server threads can use during each scheduling interval. The value is reported as a percentage of available cycles within a 10000-cycle scheduling interval. For example, the value 100 means that threads can use CPU cores are their full capacity.
cpu_affinity_mask	**bigint**	A bit mask describing which logical processors the SQL Server process can use within the processor group. For example, cpu_affinity_mask 255 (1111 1111 in binary) means that the first eight logical processors can be used.
cpu_affinity_group	**int**	The number of the processor group that is used by SQL Server.
memory_limit_mb	**bigint**	The maximum amount of committed memory, in MB, that all processes in the job object, including SQL Server, can use cumulatively.
process_memory_limit_mb	**bigint**	The maximum amount of committed memory, in MB, that a single process in the job object, such as SQL Server, can use.
workingset_limit_mb	**bigint**	The maximum amount of memory, in MB, that the SQL Server working set can use.

COLUMNS	DATA TYPE	DESCRIPTION
non_sos_mem_gap_mb	**bigint**	The amount of memory, in MB, set aside for thread stacks, DLLs, and other non-SOS memory allocations. SOS target memory is the difference between `process_memory_limit_mb` and `non_sos_mem_gap_mb`.
low_mem_signal_threshold_mb	**bigint**	A memory threshold, in MB. When the amount of available memory for the job object is below this threshold, a low memory notification signal is sent to the SQL Server process.
total_user_time	**bigint**	The total number of 100 ns ticks that threads within the job object have spent in user mode, since the job object was created.
total_kernel_time	**bigint**	The total number of 100 ns ticks that threads within the job object have spent in kernel mode, since the job object was created.
write_operation_count	**bigint**	The total number of write IO operations on local disks issued by SQL Server since the job object was created.
read_operation_count	**bigint**	The total number of read IO operations on local disks issued by SQL Server since the job object was created.
peak_process_memory_used_mb	**bigint**	The peak amount of memory, in MB, that a single process in the job object, such as SQL Server, has used since the job object was created.
peak_job_memory_used_mb	**bigint**	The peak amount of memory, in MB, that all processes in the job object have used cumulatively since the job object was created.

Permissions

On SQL Database Managed Instance, requires `VIEW SERVER STATE` permission. On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

For information on Managed Instances, see SQL Database Managed Instance.

sys.dm_os_latch_stats (Transact-SQL)

5/4/2018 • 6 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about all latch waits organized by class.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_latch_stats**.

COLUMN NAME	DATA TYPE	DESCRIPTION
latch_class	**nvarchar(120)**	Name of the latch class.
waiting_requests_count	**bigint**	Number of waits on latches in this class. This counter is incremented at the start of a latch wait.
wait_time_ms	**bigint**	Total wait time, in milliseconds, on latches in this class. **Note:** This column is updated every five minutes during a latch wait and at the end of a latch wait.
max_wait_time_ms	**bigint**	Maximum time a memory object has waited on this latch. If this value is unusually high, it might indicate an internal deadlock.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

sys.dm_os_latch_stats can be used to identify the source of latch contention by examining the relative wait numbers and wait times for the different latch classes. In some situations, you may be able to resolve or reduce latch contention. However, there might be situations that will require that you to contact Microsoft Customer Support Services.

You can reset the contents of sys.dm_os_latch_stats by using `DBCC SQLPERF` as follows:

```
DBCC SQLPERF ('sys.dm_os_latch_stats', CLEAR);
GO
```

This resets all counters to 0.

> **NOTE**
>
> These statistics are not persisted if SQL Server is restarted. All data is cumulative since the last time the statistics were reset, or since SQL Server was started.

Latches

A latch is a lightweight synchronization object that is used by various SQL Server components. A latch is primarily used to synchronize database pages. Each latch is associated with a single allocation unit.

A latch wait occurs when a latch request cannot be granted immediately, because the latch is held by another thread in a conflicting mode. Unlike locks, a latch is released immediately after the operation, even in write operations.

Latches are grouped into classes based on components and usage. Zero or more latches of a particular class can exist at any point in time in an instance of SQL Server.

> **NOTE**
>
> sys.dm_os_latch_stats does not track latch requests that were granted immediately, or that failed without waiting.

The following table contains brief descriptions of the various latch classes.

LATCH CLASS	DESCRIPTION
ALLOC_CREATE_RINGBUF	Used internally by SQL Server to initialize the synchronization of the creation of an allocation ring buffer.
ALLOC_CREATE_FREESPACE_CACHE	Used to initialize the synchronization of internal freespace caches for heaps.
ALLOC_CACHE_MANAGER	Used to synchronize internal coherency tests.
ALLOC_FREESPACE_CACHE	Used to synchronize the access to a cache of pages with available space for heaps and binary large objects (BLOBs). Contention on latches of this class can occur when multiple connections try to insert rows into a heap or BLOB at the same time. You can reduce this contention by partitioning the object. Each partition has its own latch. Partitioning will distribute the inserts across multiple latches.
ALLOC_EXTENT_CACHE	Used to synchronize the access to a cache of extents that contains pages that are not allocated. Contention on latches of this class can occur when multiple connections try to allocate data pages in the same allocation unit at the same time. This contention can be reduced by partitioning the object of which this allocation unit is a part.

LATCH CLASS	DESCRIPTION
ACCESS_METHODS_DATASET_PARENT	Used to synchronize child dataset access to the parent dataset during parallel operations.
ACCESS_METHODS_HOBT_FACTORY	Used to synchronize access to an internal hash table.
ACCESS_METHODS_HOBT	Used to synchronize access to the in-memory representation of a HoBt.
ACCESS_METHODS_HOBT_COUNT	Used to synchronize access to a HoBt page and row counters.
ACCESS_METHODS_HOBT_VIRTUAL_ROOT	Used to synchronize access to the root page abstraction of an internal B-tree.
ACCESS_METHODS_CACHE_ONLY_HOBT_ALLOC	Used to synchronize worktable access.
ACCESS_METHODS_BULK_ALLOC	Used to synchronize access within bulk allocators.
ACCESS_METHODS_SCAN_RANGE_GENERATOR	Used to synchronize access to a range generator during parallel scans.
ACCESS_METHODS_KEY_RANGE_GENERATOR	Used to synchronize access to read-ahead operations during key range parallel scans.
APPEND_ONLY_STORAGE_INSERT_POINT	Used to synchronize inserts in fast append-only storage units.
APPEND_ONLY_STORAGE_FIRST_ALLOC	Used to synchronize the first allocation for an append-only storage unit.
APPEND_ONLY_STORAGE_UNIT_MANAGER	Used for internal data structure access synchronization within the fast append-only storage unit manager.
APPEND_ONLY_STORAGE_MANAGER	Used to synchronize shrink operations in the fast append-only storage unit manager.
BACKUP_RESULT_SET	Used to synchronize parallel backup result sets.
BACKUP_TAPE_POOL	Used to synchronize backup tape pools.
BACKUP_LOG_REDO	Used to synchronize backup log redo operations.
BACKUP_INSTANCE_ID	Used to synchronize the generation of instance IDs for backup performance monitor counters.
BACKUP_MANAGER	Used to synchronize the internal backup manager.
BACKUP_MANAGER_DIFFERENTIAL	Used to synchronize differential backup operations with DBCC.
BACKUP_OPERATION	Used for internal data structure synchronization within a backup operation, such as database, log, or file backup.

LATCH CLASS	DESCRIPTION
BACKUP_FILE_HANDLE	Used to synchronize file open operations during a restore operation.
BUFFER	Used to synchronize short term access to database pages. A buffer latch is required before reading or modifying any database page. Buffer latch contention can indicate several issues, including hot pages and slow I/Os. This latch class covers all possible uses of page latches. sys.dm_os_wait_stats makes a difference between page latch waits that are caused by I/O operations and read and write operations on the page.
BUFFER_POOL_GROW	Used for internal buffer manager synchronization during buffer pool grow operations.
DATABASE_CHECKPOINT	Used to serialize checkpoints within a database.
CLR_PROCEDURE_HASHTABLE	Internal use only.
CLR_UDX_STORE	Internal use only.
CLR_DATAT_ACCESS	Internal use only.
CLR_XVAR_PROXY_LIST	Internal use only.
DBCC_CHECK_AGGREGATE	Internal use only.
DBCC_CHECK_RESULTSET	Internal use only.
DBCC_CHECK_TABLE	Internal use only.
DBCC_CHECK_TABLE_INIT	Internal use only.
DBCC_CHECK_TRACE_LIST	Internal use only.
DBCC_FILE_CHECK_OBJECT	Internal use only.
DBCC_PERF	Used to synchronize internal performance monitor counters.
DBCC_PFS_STATUS	Internal use only.
DBCC_OBJECT_METADATA	Internal use only.
DBCC_HASH_DLL	Internal use only.
EVENTING_CACHE	Internal use only.
FCB	Used to synchronize access to the file control block.
FCB_REPLICA	Internal use only.

LATCH CLASS	DESCRIPTION
FGCB_ALLOC	Use to synchronize access to round robin allocation information within a filegroup.
FGCB_ADD_REMOVE	Use to synchronize access to filegroups for ADD and DROP file operations.
FILEGROUP_MANAGER	Internal use only.
FILE_MANAGER	Internal use only.
FILESTREAM_FCB	Internal use only.
FILESTREAM_FILE_MANAGER	Internal use only.
FILESTREAM_GHOST_FILES	Internal use only.
FILESTREAM_DFS_ROOT	Internal use only.
LOG_MANAGER	Internal use only.
FULLTEXT_DOCUMENT_ID	Internal use only.
FULLTEXT_DOCUMENT_ID_TRANSACTION	Internal use only.
FULLTEXT_DOCUMENT_ID_NOTIFY	Internal use only.
FULLTEXT_LOGS	Internal use only.
FULLTEXT_CRAWL_LOG	Internal use only.
FULLTEXT_ADMIN	Internal use only.
FULLTEXT_AMDIN_COMMAND_CACHE	Internal use only.
FULLTEXT_LANGUAGE_TABLE	Internal use only.
FULLTEXT_CRAWL_DM_LIST	Internal use only.
FULLTEXT_CRAWL_CATALOG	Internal use only.
FULLTEXT_FILE_MANAGER	Internal use only.
DATABASE_MIRRORING_REDO	Internal use only.
DATABASE_MIRRORING_SERVER	Internal use only.
DATABASE_MIRRORING_CONNECTION	Internal use only.
DATABASE_MIRRORING_STREAM	Internal use only.

LATCH CLASS	DESCRIPTION
QUERY_OPTIMIZER_VD_MANAGER	Internal use only.
QUERY_OPTIMIZER_ID_MANAGER	Internal use only.
QUERY_OPTIMIZER_VIEW_REP	Internal use only.
RECOVERY_BAD_PAGE_TABLE	Internal use only.
RECOVERY_MANAGER	Internal use only.
SECURITY_OPERATION_RULE_TABLE	Internal use only.
SECURITY_OBJPERM_CACHE	Internal use only.
SECURITY_CRYPTO	Internal use only.
SECURITY_KEY_RING	Internal use only.
SECURITY_KEY_LIST	Internal use only.
SERVICE_BROKER_CONNECTION_RECEIVE	Internal use only.
SERVICE_BROKER_TRANSMISSION	Internal use only.
SERVICE_BROKER_TRANSMISSION_UPDATE	Internal use only.
SERVICE_BROKER_TRANSMISSION_STATE	Internal use only.
SERVICE_BROKER_TRANSMISSION_ERRORS	Internal use only.
SSBXmitWork	Internal use only.
SERVICE_BROKER_MESSAGE_TRANSMISSION	Internal use only.
SERVICE_BROKER_MAP_MANAGER	Internal use only.
SERVICE_BROKER_HOST_NAME	Internal use only.
SERVICE_BROKER_READ_CACHE	Internal use only.
SERVICE_BROKER_WAITFOR_MANAGER	Used to synchronize an instance level map of waiter queues. One queue exists per database ID, Database Version, and Queue ID tuple. Contention on latches of this class can occur when many connections are: In a WAITFOR(RECEIVE) wait state; calling WAITFOR(RECEIVE); exceeding the WAITFOR timeout; receiving a message; committing or rolling back the transaction that contains the WAITFOR(RECEIVE); You can reduce the contention by reducing the number of threads in a WAITFOR(RECEIVE) wait state.
SERVICE_BROKER_WAITFOR_TRANSACTION_DATA	Internal use only.

LATCH CLASS	DESCRIPTION
SERVICE_BROKER_TRANSMISSION_TRANSACTION_DATA	Internal use only.
SERVICE_BROKER_TRANSPORT	Internal use only.
SERVICE_BROKER_MIRROR_ROUTE	Internal use only.
TRACE_ID	Internal use only.
TRACE_AUDIT_ID	Internal use only.
TRACE	Internal use only.
TRACE_CONTROLLER	Internal use only.
TRACE_EVENT_QUEUE	Internal use only.
TRANSACTION_DISTRIBUTED_MARK	Internal use only.
TRANSACTION_OUTCOME	Internal use only.
NESTING_TRANSACTION_READONLY	Internal use only.
NESTING_TRANSACTION_FULL	Internal use only.
MSQL_TRANSACTION_MANAGER	Internal use only.
DATABASE_AUTONAME_MANAGER	Internal use only.
UTILITY_DYNAMIC_VECTOR	Internal use only.
UTILITY_SPARSE_BITMAP	Internal use only.
UTILITY_DATABASE_DROP	Internal use only.
UTILITY_DYNAMIC_MANAGER_VIEW	Internal use only.
UTILITY_DEBUG_FILESTREAM	Internal use only.
UTILITY_LOCK_INFORMATION	Internal use only.
VERSIONING_TRANSACTION	Internal use only.
VERSIONING_TRANSACTION_LIST	Internal use only.
VERSIONING_TRANSACTION_CHAIN	Internal use only.
VERSIONING_STATE	Internal use only.
VERSIONING_STATE_CHANGE	Internal use only.

LATCH CLASS	DESCRIPTION
KTM_VIRTUAL_CLOCK	Internal use only.

See Also

DBCC SQLPERF (Transact-SQL)

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_loaded_modules (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each module loaded into the server address space.

> **NOTE**
>
> To call this from Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_loaded_modules**.

COLUMN NAME	DATA TYPE	DESCRIPTION
base_address	**varbinary(8)**	Address of the module in the process.
file_version	**varchar(23)**	Version of the file. Appears in the following format: x.x:x.x
product_version	**varchar(23)**	Version of the product. Appears in the following format: x.x:x.x
debug	**bit**	1 = Module is a debug version of the loaded module.
patched	**bit**	1 = Module has been patched.
prerelease	**bit**	1 = Module is a pre-release version of the loaded module.
private_build	**bit**	1 = Module is a private build of the loaded module.
special_build	**bit**	1 = Module is a special build of the loaded module.
language	**int**	Language of version information of the module.
company	**nvarchar(256)**	Name of company that created the module.
description	**nvarchar(256)**	Description of the module.
name	**nvarchar(255)**	Name of module. Includes the full path of the module.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Dynamic Management Views and Functions (Transact-SQL)
SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_memory_brokers (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Allocations that are internal to SQL Server use the SQL Server memory manager. Tracking the difference between process memory counters from **sys.dm_os_process_memory** and internal counters can indicate memory use from external components in the SQL Server memory space.

Memory brokers fairly distribute memory allocations between various components within SQL Server, based on current and projected usage. Memory brokers do not perform allocations. They only track allocations for computing distribution.

The following table provides information about memory brokers.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_memory_brokers**.

COLUMN NAME	DATA TYPE	DESCRIPTION
pool_id	**int**	ID of the resource pool if it is associated with a Resource Governor pool.
memory_broker_type	**nvarchar(60)**	Type of memory broker. There are currently three types of memory brokers in SQL Server, listed below with their descriptions. **MEMORYBROKER_FOR_CACHE** : Memory that is allocated for use by cached objects. **MEMORYBROKER_FOR_STEAL** : Memory that is stolen from the buffer pool. This memory is not available for reuse by other components until it is freed by the current owner. **MEMORYBROKER_FOR_RESERVE** : Memory reserved for future use by currently executing requests.
allocations_kb	**bigint**	Amount of memory, in kilobytes (KB), that has been allocated to this type of broker.
allocations_kb_per_sec	**bigint**	Rate of memory allocations in kilobytes (KB) per second. This value can be negative for memory deallocations.

COLUMN NAME	DATA TYPE	DESCRIPTION
predicted_allocations_kb	**bigint**	Predicted amount of allocated memory by the broker. This is based on the memory usage pattern.
target_allocations_kb	**bigint**	Recommended amount of allocated memory, in kilobytes (KB), that is based on current settings and the memory usage pattern. This broker should grow to or shrink to this number.
future_allocations_kb	**bigint**	Projected number of allocations, in kilobytes (KB), that will be done in the next several seconds.
overall_limit_kb	**bigint**	Maximum amount of memory, in kilobytes (KB), that the the broker can allocate.
last_notification	**nvarchar(60)**	Memory usage recommendation that is based on the current settings and usage pattern. Valid values are as follows: grow shrink stable
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

[SQL Server Operating System Related Dynamic Management Views (Transact-SQL)]

sys.dm_os_memory_cache_clock_hands (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the status of each hand for a specific cache clock.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_memory_cache_clock_hands**.

COLUMN NAME	DATA TYPE	DESCRIPTION
cache_address	**varbinary(8)**	Address of the cache associated with the clock. Is not nullable.
name	**nvarchar(256)**	Name of the cache. Is not nullable.
type	**nvarchar(60)**	Type of cache store. There can be several caches of the same type. Is not nullable.
clock_hand	**nvarchar(60)**	Type of hand. This is one of the following: External Internal Is not nullable.
clock_status	**nvarchar(60)**	Status of the clock. This is one of the following: Suspended Running Is not nullable.
rounds_count	**bigint**	Number of sweeps made through the cache to remove entries. Is not nullable.
removed_all_rounds_count	**bigint**	Number of entries removed by all sweeps. Is not nullable.
updated_last_round_count	**bigint**	Number of entries updated during the last sweep. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
removed_last_round_count	**bigint**	Number of entries removed during the last sweep. Is not nullable.
last_tick_time	**bigint**	Last time, in milliseconds, that the clock hand moved. Is not nullable.
round_start_time	**bigint**	Time, in milliseconds, of the previous sweep. Is not nullable.
last_round_start_time	**bigint**	Total time, in milliseconds, taken by the clock to complete the previous round. Is not nullable.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

SQL Server stores information in memory in a structure called a memory cache. The information in the cache can be data, index entries, compiled procedure plans, and a variety of other types of SQL Server information. To avoid re-creating the information, it is retained the memory cache as long as possible and is ordinarily removed from the cache when it is too old to be useful, or when the memory space is needed for new information. The process that removes old information is called a memory sweep. The memory sweep is a frequent activity, but is not continuous. A clock algorithm controls the sweep of the memory cache. Each clock can control several memory sweeps, which are called hands. The memory-cache clock hand is the current location of one of the hands of a memory sweep.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)
sys.dm_os_memory_cache_counters (Transact-SQL)

sys.dm_os_memory_cache_counters (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a snapshot of the health of a cache in SQL Server. **sys.dm_os_memory_cache_counters** provides run-time information about the cache entries allocated, their use, and the source of memory for the cache entries.

> **NOTE:** To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_memory_cache_counters**.

COLUMN NAME	DATA TYPE	DESCRIPTION
cache_address	**varbinary(8)**	Indicates the address (primary key) of the counters associated with a specific cache. Is not nullable.
name	**nvarchar(256)**	Specifies the name of the cache. Is not nullable.
type	**nvarchar(60)**	Indicates the type of cache that is associated with this entry. Is not nullable.
single_pages_kb	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Amount, in kilobytes, of the single-page memory allocated. This is the amount of memory allocated by using the single-page allocator. This refers to the 8-KB pages that are taken directly from the buffer pool for this cache. Is not nullable.
pages_kb	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Specifies the amount, in kilobytes, of the memory allocated in the cache. Is not nullable.
multi_pages_kb	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Amount, in kilobytes, of the multipage memory allocated. This is the amount of memory allocated by using the multiple-page allocator of the memory node. This memory is allocated outside the buffer pool and takes advantage of the virtual allocator of the memory nodes. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
pages_in_use_kb	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Specifies the amount, in kilobytes, of the memory that is allocated and in use in the cache. Is nullable. Values for objects of type `USERSTORE_<*>` are not tracked. NULL is reported for them.
single_pages_in_use_kb	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Amount, in kilobytes, of the single-page memory that is being used. Is nullable. This information is not tracked for objects of type USERSTORE_<*> and these values will be NULL.
multi_pages_in_use_kb	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Amount, in kilobytes, of the multipage memory that is being used. NULLABLE. This information is not tracked for objects of type USERSTORE_<*>, and these values will be NULL.
entries_count	**bigint**	Indicates the number of entries in the cache. Is not nullable.
entries_in_use_count	**bigint**	Indicates the number of entries in the cache that is being used. Is not nullable.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_memory_cache_entries (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about all entries in caches in SQL Server. Use this view to trace cache entries to their associated objects. You can also use this view to obtain statistics on cache entries.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_memory_cache_entries**.

COLUMN NAME	DATA TYPE	DESCRIPTION
cache_address	**varbinary(8)**	Address of the cache. Is not nullable.
name	**nvarchar(256)**	Name of the cache. Is not nullable.
type	**varchar(60)**	Type of cache. Is not nullable.
entry_address	**varbinary(8)**	Address of the descriptor of the cache entry. Is not nullable.
entry_data_address	**varbinary(8)**	Address of the user data in the cache entry. 0x00000000 = Entry data address is not available. Is not nullable.
in_use_count	**int**	Number of concurrent users of this cache entry. Is not nullable.
is_dirty	**bit**	Indicates whether this cache entry is marked for removal. 1 = marked for removal. Is not nullable.
disk_ios_count	**int**	Number of I/Os incurred while this entry was created. Is not nullable.
context_switches_count	**int**	Number of context switches incurred while this entry was created. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
original_cost	**int**	Original cost of the entry. This value is an approximation of the number of I/Os incurred, CPU instruction cost, and the amount of memory consumed by entry. The greater the cost, the lower the chance that the item will be removed from the cache. Is not nullable.
current_cost	**int**	Current cost of the cache entry. This value is updated during the process of entry purging. Current cost is reset to its original value on entry reuse. Is not nullable.
memory_object_address	**varbinary(8)**	Address of the associated memory object. Is nullable.
pages_allocated_count	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Number of 8-KB pages to store this cache entry. Is not nullable.
pages_kb	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Amount of memory in kilobytes (KB) used by this cache entry. Is not nullable.
entry_data	**nvarchar(2048)**	Serialized representation of the cached entry. This information is cache store dependant. Is nullable.
pool_id	**int**	**Applies to**: SQL Server 2008 R2 through SQL Server 2017. Resource pool id associated with entry. Is nullable. not katmai
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_memory_cache_hash_tables (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each active cache in the instance of SQL Server.

NOTE

To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_memory_cache_hash_tables**.

COLUMN NAME	DATA TYPE	DESCRIPTION
cache_address	**varbinary(8)**	Address (primary key) of the cache entry. Is not nullable.
name	**nvarchar(256)**	Name of the cache. Is not nullable.
type	**nvarchar(60)**	Type of cache. Is not nullable.
table_level	**int**	Hash table number. A particular cache may have multiple hash tables that correspond to different hash functions. Is not nullable.
buckets_count	**int**	Number of buckets in the hash table. Is not nullable.
buckets_in_use_count	**int**	Number of buckets that are currently being used. Is not nullable.
buckets_min_length	**int**	Minimum number of cache entries in a bucket. Is not nullable.
buckets_max_length	**int**	Maximum number of cache entries in a bucket. Is not nullable.
buckets_avg_length	**int**	Average number of cache entries in each bucket. Is not nullable.
buckets_max_length_ever	**int**	Maximum number of cached entries in a hash bucket for this hash table since the server was started. Is not nullable.
hits_count	**bigint**	Number of cache hits. Is not nullable.
misses_count	**bigint**	Number of cache misses. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
buckets_avg_scan_hit_length	**int**	Average number of examined entries in a bucket before the searched for an item was found. Is not nullable.
buckets_avg_scan_miss_length	**int**	Average number of examined entries in a bucket before the search ended unsuccessfully. Is not nullable.
pdw_node_id	**int**	The identifier for the node that this distribution is on. **Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_memory_clerks (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the set of all memory clerks that are currently active in the instance of SQL Server.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_memory_clerks**.

COLUMN NAME	DATA TYPE	DESCRIPTION
memory_clerk_address	**varbinary(8)**	Specifies the unique memory address of the memory clerk. This is the primary key column. Is not nullable.
type	**nvarchar(60)**	Specifies the type of memory clerk. Every clerk has a specific type, such as CLR Clerks MEMORYCLERK_SQLCLR. Is not nullable.
name	**nvarchar(256)**	Specifies the internally assigned name of this memory clerk. A component can have several memory clerks of a specific type. A component might choose to use specific names to identify memory clerks of the same type. Is not nullable.
memory_node_id	**smallint**	Specifies the ID of the memory node. Not nullable.
single_pages_kb	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2.
pages_kb	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Specifies the amount of page memory allocated in kilobytes (KB) for this memory clerk. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
multi_pages_kb	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Amount of multipage memory allocated in KB. This is the amount of memory allocated by using the multiple page allocator of the memory nodes. This memory is allocated outside the buffer pool and takes advantage of the virtual allocator of the memory nodes. Is not nullable.
virtual_memory_reserved_kb	**bigint**	Specifies the amount of virtual memory that is reserved by a memory clerk. Is not nullable.
virtual_memory_committed_kb	**bigint**	Specifies the amount of virtual memory that is committed by a memory clerk. The amount of committed memory should always be less than the amount of reserved memory. Is not nullable.
awe_allocated_kb	**bigint**	Specifies the amount of memory in kilobytes (KB) locked in the physical memory and not paged out by the operating system. Is not nullable.
shared_memory_reserved_kb	**bigint**	Specifies the amount of shared memory that is reserved by a memory clerk. The amount of memory reserved for use by shared memory and file mapping. Is not nullable.
shared_memory_committed_kb	**bigint**	Specifies the amount of shared memory that is committed by the memory clerk. Is not nullable.
page_size_in_bytes	**bigint**	Specifies the granularity of the page allocation for this memory clerk. Is not nullable.
page_allocator_address	**varbinary(8)**	Specifies the address of the page allocator. This address is unique for a memory clerk and can be used in **sys.dm_os_memory_objects** to locate memory objects that are bound to this clerk. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
host_address	**varbinary(8)**	Specifies the memory address of the host for this memory clerk. For more information, see sys.dm_os_hosts (Transact-SQL). Components, such as Microsoft SQL Server Native Client, access SQL Server memory resources through the host interface. 0x00000000 = Memory clerk belongs to SQL Server. Is not nullable.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

The SQL Server memory manager consists of a three-layer hierarchy. At the bottom of the hierarchy are memory nodes. The middle level consists of memory clerks, memory caches, and memory pools. The top layer consists of memory objects. These objects are generally used to allocate memory in an instance of SQL Server.

Memory nodes provide the interface and the implementation for low-level allocators. Inside SQL Server, only memory clerks have access to memory nodes. Memory clerks access memory node interfaces to allocate memory. Memory nodes also track the memory allocated by using the clerk for diagnostics. Every component that allocates a significant amount of memory must create its own memory clerk and allocate all its memory by using the clerk interfaces. Frequently, components create their corresponding clerks at the time SQL Server is started.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)
sys.dm_os_sys_info (Transact-SQL)
sys.dm_exec_query_memory_grants (Transact-SQL)
sys.dm_exec_requests (Transact-SQL)
sys.dm_exec_query_plan (Transact-SQL)
sys.dm_exec_sql_text (Transact-SQL)

sys.dm_os_memory_nodes (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Allocations that are internal to SQL Server use the SQL Server memory manager. Tracking the difference between process memory counters from **sys.dm_os_process_memory** and internal counters can indicate memory use from external components in the SQL Server memory space.

Nodes are created per physical NUMA memory nodes. These might be different from the CPU nodes in **sys.dm_os_nodes**.

No allocations done directly through Windows memory allocations routines are tracked. The following table provides information about memory allocations done only by using SQL Server memory manager interfaces.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_memory_nodes**.

COLUMN NAME	DATA TYPE	DESCRIPTION
memory_node_id	**smallint**	Specifies the ID of the memory node. Related to **memory_node_id** of **sys.dm_os_memory_clerks**. Not nullable.
virtual_address_space_reserved_kb	**bigint**	Indicates the number of virtual address reservations, in kilobytes (KB), which are neither committed nor mapped to physical pages. Not nullable.
virtual_address_space_committed_kb	**bigint**	Specifies the amount of virtual address, in KB, that has been committed or mapped to physical pages. Not nullable.
locked_page_allocations_kb	**bigint**	Specifies the amount of physical memory, in KB, that has been locked by SQL Server. Not nullable.
single_pages_kb	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Amount of committed memory, in KB, that is allocated by using the single page allocator by threads running on this node. This memory is allocated from the buffer pool. This value indicates the node where allocations request occurred, not the physical location where the allocation request was satisfied.

COLUMN NAME	DATA TYPE	DESCRIPTION
pages_kb	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Specifies the amount of committed memory, in KB, which is allocated from this NUMA node by Memory Manager Page Allocator. Not nullable.
multi_pages_kb	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Amount of committed memory, in KB, that is allocated by using the multipage allocator by threads running on this node. This memory is from outside the buffer pool. This value indicates the node where the allocation requests occurred, not the physical location where the allocation request was satisfied.
shared_memory_reserved_kb	**bigint**	Specifies the amount of shared memory, in KB, that has been reserved from this node. Not nullable.
shared_memory_committed_kb	**bigint**	Specifies the amount of shared memory, in KB, that has been committed on this node. Not nullable.
cpu_affinity_mask	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Internal use only. Not nullable.
online_scheduler_mask	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Internal use only. Not nullable.
processor_group	**smallint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Internal use only. Not nullable.
foreign_committed_kb	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Specifies the amount of committed memory, in KB, from other memory nodes. Not nullable.
target_kb	**bigint**	**Applies to**: SQL Server 2016 (13.x) through SQL Server 2017, SQL Database. Specifies the memory goal for the memory node, in KB.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_memory_objects (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns memory objects that are currently allocated by SQL Server. You can use **sys.dm_os_memory_objects** to analyze memory use and to identify possible memory leaks.

COLUMN NAME	DATA TYPE	DESCRIPTION
memory_object_address	**varbinary(8)**	Address of the memory object. Is not nullable.
parent_address	**varbinary(8)**	Address of the parent memory object. Is nullable.
pages_allocated_count	**int**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Number of pages that are allocated by this object. Is not nullable.
pages_in_bytes	**bigint**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Amount of memory in bytes that is allocated by this instance of the memory object. Is not nullable.
creation_options	**int**	Internal use only. Is nullable.
bytes_used	**bigint**	Internal use only. Is nullable.
type	**nvarchar(60)**	Type of memory object. This indicates some component that this memory object belongs to, or the function of the memory object. Is nullable.
name	**varchar(128)**	Internal use only. Nullable.
memory_node_id	**smallint**	ID of a memory node that is being used by this memory object. Is not nullable.
creation_time	**datetime**	Internal use only. Is nullable.
max_pages_allocated_count	**int**	**Applies to**: SQL Server 2008 through SQL Server 2008 R2. Maximum number of pages allocated by this memory object. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
page_size_in_bytes	**int**	**Applies to**: SQL Server 2012 (11.x) through SQL Server 2017. Size of pages in bytes allocated by this object. Is not nullable.
max_pages_in_bytes	**bigint**	Maximum amount of memory ever used by this memory object. Is not nullable.
page_allocator_address	**varbinary(8)**	Memory address of page allocator. Is not nullable. For more information, see sys.dm_os_memory_clerks (Transact-SQL).
creation_stack_address	**varbinary(8)**	Internal use only. Is nullable.
sequence_num	**int**	Internal use only. Is nullable.
partition_type	**int**	The type of partition: 0 - Non-partitionable memory object 1 - Partitionable memory object, currently not partitioned 2 - Partitionable memory object, partitioned by NUMA node. In an environment with a single NUMA node this is equivalent to 1. 3 - Partitionable memory object, partitioned by CPU.
contention_factor	**real**	A value specifying contention on this memory object, with 0 meaning no contention. The value is updated whenever a specified number of memory allocations were made reflecting contention during that period. Applies only to thread-safe memory objects.
waiting_tasks_count	**bigint**	Number of waits on this memory object. This counter is incremented whenever memory is allocated from this memory object. The increment is the number of tasks currently waiting for access to this memory object. Applies only to thread-safe memory objects. This is a best effort value without a correctness guarantee.
exclusive_access_count	**bigint**	Specifies how often this memory object was accessed exclusively. Applies only to thread-safe memory objects. This is a best effort value without a correctness guarantee.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

partition_type, **contention_factor**, **waiting_tasks_count**, and **exclusive_access_count** are not yet implemented in SQL Database.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

Memory objects are heaps. They provide allocations that have a finer granularity than those provided by memory clerks. SQL Server components use memory objects instead of memory clerks. Memory objects use the page allocator interface of the memory clerk to allocate pages. Memory objects do not use virtual or shared memory interfaces. Depending on the allocation patterns, components can create different types of memory objects to allocate regions of arbitrary size.

The typical page size for a memory object is 8 KB. However, incremental memory objects can have page sizes that range from 512 bytes to 8 KB.

> **NOTE**
>
> Page size is not a maximum allocation. Instead, page size is allocation granularity that is supported by a page allocator and that is implemented by a memory clerk. You can request allocations greater than 8 KB from memory objects.

Examples

The following example returns the amount of memory allocated by each memory object type.

```
SELECT SUM (pages_in_bytes) as 'Bytes Used', type
FROM sys.dm_os_memory_objects
GROUP BY type
ORDER BY 'Bytes Used' DESC;
GO
```

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)
sys.dm_os_memory_clerks (Transact-SQL)

sys.dm_os_memory_pools (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for each object store in the instance of SQL Server. You can use this view to monitor cache memory use and to identify bad caching behavior

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_memory_pools**.

COLUMN NAME	DATA TYPE	DESCRIPTION
memory_pool_address	**varbinary(8)**	Memory address of the entry that represents the memory pool. Is not nullable.
pool_id	**int**	ID of a specific pool within a set of pools. Is not nullable.
type	**nvarchar(60)**	Type of object pool. Is not nullable. For more information, see sys.dm_os_memory_clerks (Transact-SQL).
name	**nvarchar(256)**	System-assigned name of this memory object. Is not nullable.
max_free_entries_count	**bigint**	Maximum number of free entries that a pool can have. Is not nullable.
free_entries_count	**bigint**	Number of free entries currently in the pool. Is not nullable.
removed_in_all_rounds_count	**bigint**	Number of entries removed from the pool since the instance of SQL Server was started. Is not nullable.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

SQL Server components sometimes use a common pool framework to cache homogeneous, stateless types of data. The pool framework is simpler than cache framework. All entries in the pools are considered equal. Internally, pools are memory clerks and can be used in places where memory clerks are used.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_nodes (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

An internal component named the SQLOS creates node structures that mimic hardware processor locality. These structures can be changed by using soft-NUMA to create custom node layouts.

> **NOTE**
>
> Starting with SQL Server 2016 (13.x), the SQL Server Database Engine will automatically use soft-NUMA for certain hardware configurations. For more information, see Automatic Soft-NUMA.

The following table provides information about these nodes.

> **NOTE**
>
> To call this DMV from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_nodes**.

COLUMN NAME	DATA TYPE	DESCRIPTION
node_id	**smallint**	ID of the node.

COLUMN NAME	DATA TYPE	DESCRIPTION
node_state_desc	**nvarchar(256)**	Description of the node state. Values are displayed with the mutually exclusive values first, followed by the combinable values. For example: Online, Thread Resources Low, Lazy Preemptive There are four mutually exclusive node_state_desc values. They are listed below with their descriptions. • ONLINE: Node is online • OFFLINE: Node is offline • IDLE: Node has no pending work requests, and has entered an idle state. • IDLE_READY: Node has no pending work requests, and is ready to enter an idle state. There are three combinable node_state_desc values, listed below with their descriptions. • DAC: This node is reserved for the Dedicated Administrative Connection. • THREAD_RESOURCES_LOW: No new threads can be created on this node because of a low-memory condition. • HOT ADDED: Indicates the nodes were added in response to a hot add CPU event.
memory_object_address	**varbinary(8)**	Address of memory object associated with this node. One-to-one relation to sys.dm_os_memory_objects.memory_object_address.
memory_clerk_address	**varbinary(8)**	Address of memory clerk associated with this node. One-to-one relation to sys.dm_os_memory_clerks.memory_clerk_address.
io_completion_worker_address	**varbinary(8)**	Address of worker assigned to IO completion for this node. One-to-one relation to sys.dm_os_workers.worker_address.
memory_node_id	**smallint**	ID of the memory node this node belongs to. Many-to-one relation to sys.dm_os_memory_nodes.memory_node_id.
cpu_affinity_mask	**bigint**	Bitmap identifying the CPUs this node is associated with.

COLUMN NAME	DATA TYPE	DESCRIPTION
online_scheduler_count	**smallint**	Number of online schedulers that are managed by this node.
idle_scheduler_count	**smallint**	Number of online schedulers that have no active workers.
active_worker_count	**int**	Number of workers that are active on all schedulers managed by this node.
avg_load_balance	**int**	Average number of tasks per scheduler on this node.
timer_task_affinity_mask	**bigint**	Bitmap identifying the schedulers that can have timer tasks assigned to them.
permanent_task_affinity_mask	**bigint**	Bitmap identifying the schedulers that can have permanent tasks assigned to them.
resource_monitor_state	**bit**	Each node has one resource monitor assigned to it. The resource monitor can be running or idle. A value of 1 indicates running, a value of 0 indicates idle.
online_scheduler_mask	**bigint**	Identifies the process affinity mask for this node.
processor_group	**smallint**	Identifies the group of processors for this node.
cpu_count	**int**	Number of CPUs available for this node.
pdw_node_id	**int**	The identifier for the node that this distribution is on. **Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)
Soft-NUMA (SQL Server)

sys.dm_os_performance_counters (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row per performance counter maintained by the server. For information about each performance counter, see Use SQL Server Objects.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_performance_counters**.

COLUMN NAME	DATA TYPE	DESCRIPTION
object_name	**nchar(128)**	Category to which this counter belongs.
counter_name	**nchar(128)**	Name of the counter. To get more information about a counter, this is the name of the topic to select from the list of counters in Use SQL Server Objects.
instance_name	**nchar(128)**	Name of the specific instance of the counter. Often contains the database name.
cntr_value	**bigint**	Current value of the counter. **Note:** For per-second counters, this value is cumulative. The rate value must be calculated by sampling the value at discrete time intervals. The difference between any two successive sample values is equal to the rate for the time interval used.
cntr_type	**int**	Type of counter as defined by the Windows performance architecture. See WMI Performance Counter Types on MSDN or your Windows Server documentation for more information on performance counter types.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Remarks

If the installation instance of SQL Server fails to display the performance counters of the Windows operating system, use the following Transact-SQL query to confirm that performance counters have been disabled.

```
SELECT COUNT(*) FROM sys.dm_os_performance_counters;
```

If the return value is 0 rows, this means that the performance counters have been disabled. You should then look at the setup log and search for error 3409, "Reinstall sqlctr.ini for this instance, and ensure that the instance login account has correct registry permissions." This denotes that performance counters were not enabled. The errors immediately before the 3409 error should indicate the root cause for the failure of performance counter enabling. For more information about setup log files, see View and Read SQL Server Setup Log Files.

Permission

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example returns performance counter values.

```
SELECT object_name, counter_name, instance_name, cntr_value, cntr_type
FROM sys.dm_os_performance_counters;
```

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)
sys.sysperfinfo (Transact-SQL)

sys.dm_os_process_memory (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Most memory allocations that are attributed to the SQL Server process space are controlled through interfaces that allow for tracking and accounting of those allocations. However, memory allocations might be performed in the SQL Server address space that bypasses internal memory management routines. Values are obtained through calls to the base operating system. They are not manipulated by methods internal to SQL Server, except when it adjusts for locked or large page allocations.

All returned values that indicate memory sizes are shown in kilobytes (KB). The column **total_virtual_address_space_reserved_kb** is a duplicate of **virtual_memory_in_bytes** from **sys.dm_os_sys_info**.

The following table provides a complete picture of the process address space.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_process_memory**.

COLUMN NAME	DATA TYPE	DESCRIPTION
physical_memory_in_use_kb	**bigint**	Indicates the process working set in KB, as reported by operating system, as well as tracked allocations by using large page APIs. Not nullable.
large_page_allocations_kb	**bigint**	Specifies physical memory allocated by using large page APIs. Not nullable.
locked_page_allocations_kb	**bigint**	Specifies memory pages locked in memory. Not nullable.
total_virtual_address_space_kb	**bigint**	Indicates the total size of the user mode part of the virtual address space. Not nullable.
virtual_address_space_reserved_kb	**bigint**	Indicates the total amount of virtual address space reserved by the process. Not nullable.
virtual_address_space_committed_kb	**bigint**	Indicates the amount of reserved virtual address space that has been committed or mapped to physical pages. Not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
virtual_address_space_available_kb	**bigint**	Indicates the amount of virtual address space that is currently free. Not nullable. **Note:** Free regions that are smaller than the allocation granularity can exist. These regions are unavailable for allocations.
page_fault_count	**bigint**	Indicates the number of page faults that are incurred by the SQL Server process. Not nullable.
memory_utilization_percentage	**int**	Specifies the percentage of committed memory that is in the working set. Not nullable.
available_commit_limit_kb	**bigint**	Indicates the amount of memory that is available to be committed by the process. Not nullable.
process_physical_memory_low	**bit**	Indicates that the process is responding to low physical memory notification. Not nullable.
process_virtual_memory_low	**bit**	Indicates that low virtual memory condition has been detected. Not nullable.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server requires VIEW SERVER STATE permission on the server.

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

Dynamic Management Views and Functions (Transact-SQL)
SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_schedulers (Transact-SQL)

5/4/2018 • 8 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row per scheduler in SQL Server where each scheduler is mapped to an individual processor. Use this view to monitor the condition of a scheduler or to identify runaway tasks.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_schedulers**.

COLUMN NAME	DATA TYPE	DESCRIPTION
scheduler_address	**varbinary(8)**	Memory address of the scheduler. Is not nullable.
parent_node_id	**int**	ID of the node that the scheduler belongs to, also known as the parent node. This represents a nonuniform memory access (NUMA) node. Is not nullable.
scheduler_id	**int**	ID of the scheduler. All schedulers that are used to run regular queries have ID numbers less than 1048576. Those schedulers that have IDs greater than or equal to 1048576 are used internally by SQL Server, such as the dedicated administrator connection scheduler. Is not nullable.
cpu_id	**smallint**	CPU ID assigned to the scheduler. Is not nullable. **Note:** 255 does not indicate no affinity as it did in SQL Server 2005. See sys.dm_os_threads (Transact-SQL) for additional affinity information.

COLUMN NAME	DATA TYPE	DESCRIPTION
status	**nvarchar(60)**	Indicates the status of the scheduler. Can be one of the following values: - HIDDEN ONLINE - HIDDEN OFFLINE - VISIBLE ONLINE - VISIBLE OFFLINE - VISIBLE ONLINE (DAC) - HOT_ADDED Is not nullable. HIDDEN schedulers are used to process requests that are internal to the Database Engine. VISIBLE schedulers are used to process user requests. OFFLINE schedulers map to processors that are offline in the affinity mask and are, therefore, not being used to process any requests. ONLINE schedulers map to processors that are online in the affinity mask and are available to process threads. DAC indicates the scheduler is running under a dedicated administrator connection. HOT ADDED indicates the schedulers were added in response to a hot add CPU event.
is_online	**bit**	If SQL Server is configured to use only some of the available processors on the server, this configuration can mean that some schedulers are mapped to processors that are not in the affinity mask. If that is the case, this column returns 0. This value means that the scheduler is not being used to process queries or batches. Is not nullable.
is_idle	**bit**	1 = Scheduler is idle. No workers are currently running. Is not nullable.
preemptive_switches_count	**int**	Number of times that workers on this scheduler have switched to the preemptive mode. To execute code that is outside SQL Server (for example, extended stored procedures and distributed queries), a thread has to execute outside the control of the non-preemptive scheduler. To do this, a worker switches to preemptive mode.

COLUMN NAME	DATA TYPE	DESCRIPTION
context_switches_count	**int**	Number of context switches that have occurred on this scheduler. Is not nullable. To allow for other workers to run, the current running worker has to relinquish control of the scheduler or switch context. **Note:** If a worker yields the scheduler and puts itself into the runnable queue and then finds no other workers, the worker will select itself. In this case, the context_switches_count is not updated, but the yield_count is updated.
idle_switches_count	**int**	Number of times the scheduler has been waiting for an event while idle. This column is similar to context_switches_count. Is not nullable.
current_tasks_count	**int**	Number of current tasks that are associated with this scheduler. This count includes the following: - Tasks that are waiting for a worker to execute them. - Tasks that are currently waiting or running (in SUSPENDED or RUNNABLE state). When a task is completed, this count is decremented. Is not nullable.
runnable_tasks_count	**int**	Number of workers, with tasks assigned to them, that are waiting to be scheduled on the runnable queue. Is not nullable.
current_workers_count	**int**	Number of workers that are associated with this scheduler. This count includes workers that are not assigned any task. Is not nullable.
active_workers_count	**int**	Number of workers that are active. An active worker is never preemptive, must have an associated task, and is either running, runnable, or suspended. Is not nullable.
work_queue_count	**bigint**	Number of tasks in the pending queue. These tasks are waiting for a worker to pick them up. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
pending_disk_io_count	**int**	Number of pending I/Os that are waiting to be completed. Each scheduler has a list of pending I/Os that are checked to determine whether they have been completed every time there is a context switch. The count is incremented when the request is inserted. This count is decremented when the request is completed. This number does not indicate the state of the I/Os. Is not nullable.
load_factor	**int**	Internal value that indicates the perceived load on this scheduler. This value is used to determine whether a new task should be put on this scheduler or another scheduler. This value is useful for debugging purposes when it appears that schedulers are not evenly loaded. The routing decision is made based on the load on the scheduler. SQL Server also uses a load factor of nodes and schedulers to help determine the best location to acquire resources. When a task is enqueued, the load factor is increased. When a task is completed, the load factor is decreased. Using the load factors helps SQL Server OS balance the work load better. Is not nullable.
yield_count	**int**	Internal value that is used to indicate progress on this scheduler. This value is used by the Scheduler Monitor to determine whether a worker on the scheduler is not yielding to other workers on time. This value does not indicate that the worker or task transitioned to a new worker. Is not nullable.
last_timer_activity	**bigint**	In CPU ticks, the last time that the scheduler timer queue was checked by the scheduler. Is not nullable.
failed_to_create_worker	**bit**	Set to 1 if a new worker could not be created on this scheduler. This generally occurs because of memory constraints. Is nullable.
active_worker_address	**varbinary(8)**	Memory address of the worker that is currently active. Is nullable. For more information, see sys.dm_os_workers (Transact-SQL).
memory_object_address	**varbinary(8)**	Memory address of the scheduler memory object. Not NULLABLE.

COLUMN NAME	DATA TYPE	DESCRIPTION
task_memory_object_address	**varbinary(8)**	Memory address of the task memory object. Is not nullable. For more information, see sys.dm_os_memory_objects (Transact-SQL).
quantum_length_us	**bigint**	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. Exposes the scheduler quantum used by SQLOS.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

A. Monitoring hidden and nonhidden schedulers

The following query outputs the state of workers and tasks in SQL Server across all schedulers. This query was executed on a computer system that has the following:

- Two processors (CPUs)
- Two (NUMA) nodes
- One CPU per NUMA node
- Affinity mask set to `0x03`.

```
SELECT
    scheduler_id,
    cpu_id,
    parent_node_id,
    current_tasks_count,
    runnable_tasks_count,
    current_workers_count,
    active_workers_count,
    work_queue_count
  FROM sys.dm_os_schedulers;
```

Here is the result set.

```
scheduler_id cpu_id parent_node_id current_tasks_count
------------ ------ -------------- -------------------
0            1      0              9
257          255    0              1
1            0      1              10
258          255    1              1
255          255    32             2

runnable_tasks_count current_workers_count
-------------------- ---------------------
0                    11
0                    1
0                    18
0                    1
0                    3

active_workers_count work_queue_count
-------------------- --------------------
6                    0
1                    0
8                    0
1                    0
1                    0
```

The output provides the following information:

- There are five schedules. Two schedulers have an ID value < 1048576. Schedulers with ID >= 1048576are known as hidden schedulers. Scheduler `255` represents the dedicated administrator connection (DAC). There is one DAC scheduler per instance. Resource monitors that coordinate memory pressure use scheduler `257` and scheduler `258`, one per NUMA node
- There are 23 active tasks in the output. These tasks include user requests in addition to resource management tasks that have been started by SQL Server. Examples of SQL Server tasks are RESOURCE MONITOR (one per NUMA node), LAZY WRITER (one per NUMA node), LOCK MONITOR, CHECKPOINT, and LOG WRITER.
- NUMA node `0` is mapped to CPU `1` and NUMA node `1` is mapped to CPU `0`. SQL Server typically starts on a NUMA node other than node 0.
- With `runnable_tasks_count` returning `0`, there are no actively running tasks. However, active sessions may exist.
- Scheduler `255` representing DAC has `3` workers associated with it. These workers are allocated at SQL Server startup and do not change. These workers are used to process DAC queries only. The two tasks on this scheduler represent a connection manager and an idle worker.
- `active_workers_count` represents all workers that have associated tasks and are running under non-preemptive mode. Some tasks, such as network listeners, run under preemptive scheduling.
- Hidden schedulers do not process typical user requests. The DAC scheduler is the exception. This DAC scheduler has one thread to process requests.

B. Monitoring nonhidden schedulers in a busy system

The following query shows the state of heavily loaded nonhidden schedulers, where more requests exist than can be handled by available workers. In this example, 256 workers are assigned tasks. Some tasks are waiting for an assignment to a worker. Lower runnable count implies that multiple tasks are waiting for a resource.

> **NOTE**
>
> You can find the state of workers by querying sys.dm_os_workers. For more information, see sys.dm_os_workers (Transact-SQL).

Here is the query:

```
SELECT
    scheduler_id,
    cpu_id,
    current_tasks_count,
    runnable_tasks_count,
    current_workers_count,
    active_workers_count,
    work_queue_count
  FROM sys.dm_os_schedulers
  WHERE scheduler_id < 255;
```

Here is the result set.

```
scheduler_id current_tasks_count runnable_tasks_count
------------ ------------------- --------------------
0            144                 0
1            147                 1

current_workers_count active_workers_count work_queue_count
--------------------- -------------------- --------------------
128                   125                  16
128                   126                  19
```

By comparison, the following result shows multiple runnable tasks where no task is waiting to obtain a worker. The `work_queue_count` is `0` for both schedulers.

```
scheduler_id current_tasks_count runnable_tasks_count
------------ ------------------- --------------------
0            107                 98
1            110                 100

current_workers_count active_workers_count work_queue_count
--------------------- -------------------- --------------------
128                   104                  0
128                   108                  0
```

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_server_diagnostics_log_configurations

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row with the current configuration for the SQL Server failover cluster diagnostic log. These property settings determine whether the diagnostic logging is on or off, and the location, number, and size of the log files.

COLUMN NAME	DATA TYPE	DESCRIPTION
is_enabled	**bit**	Indicates if the logging is turned on or off. 1 = Diagnostics logging is turned on 0 = Diagnostics logging is turned off
max_size	**int**	Maximum size in megabytes to which each of the diagnostic logs can grow. The default is 100 MB.
max_files	**int**	Maximum number of diagnostic log files that can be stored on the computer before they are recycled for new diagnostic logs.
path	**nvarchar(260)**	Path indicating the location of the diagnostic logs. The default location is <\MSSQL\Log> within the installation folder of the SQL Server failover cluster instance.

Permissions

Requires VIEW SERVER STATE permissions on the SQL Server failover cluster instance.

Examples

The following example uses sys.dm_os_server_diagnostics_log_configurations to return the property settings for the SQL Server failover diagnostic logs.

```
SELECT <list of columns>
FROM sys.dm_os_server_diagnostics_log_configurations;
```

Here is the result set.

IS_ENABLED	PATH	MAX_SIZE	MAX_FILES

IS_ENABLED	PATH	MAX_SIZE	MAX_FILES
1	<C:\Program Files\Microsoft SQL Server\MSSQL13\MSSQL\Log>	10	10

See Also

View and Read Failover Cluster Instance Diagnostics Log

sys.dm_os_stacks (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This dynamic management view is used internally by SQL Server to do the following:

- Keep track of debug data such as outstanding allocations.
- Assume or validate logic that is used by SQL Server components in places where the component assumes that a certain call has been made.

COLUMN NAME	DATA TYPE	DESCRIPTION
stack_address	**varbinary(8)**	Unique address for this stack allocation. Is not nullable.
frame_index	**int**	Each line represents a function call that, when sorted in ascending order by frame index for a particular **stack_address**, returns the full call stack. Is not nullable.
frame_address	**varbinary(8)**	Address of the function call. Is not nullable.

Remarks

sys.dm_os_stacks requires that the symbols of the server and other components be present on the server to display the information correctly.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_sys_info (Transact-SQL)

5/4/2018 • 7 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a miscellaneous set of useful information about the computer, and about the resources available to and consumed by SQL Server.

> **NOTE:** To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_sys_info**.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC NOTES
cpu_ticks	**bigint**	Specifies the current CPU tick count. CPU ticks are obtained from the processor's RDTSC counter. It is a monotonically increasing number. Not nullable.
ms_ticks	**bigint**	Specifies the number of milliseconds since the computer started. Not nullable.
cpu_count	**int**	Specifies the number of logical CPUs on the system. Not nullable.
hyperthread_ratio	**int**	Specifies the ratio of the number of logical or physical cores that are exposed by one physical processor package. Not nullable.
physical_memory_in_bytes	**bigint**	**Applies to:** SQL Server 2008 through SQL Server 2008 R2. Specifies the total amount of physical memory on the machine. Not nullable.
physical_memory_kb	**bigint**	**Applies to:** SQL Server 2012 (11.x) through SQL Server 2017. Specifies the total amount of physical memory on the machine. Not nullable.
virtual_memory_in_bytes	**bigint**	**Applies to:** SQL Server 2008 through SQL Server 2008 R2. Amount of virtual memory available to the process in user mode. This can be used to determine whether SQL Server was started by using a 3-GB switch.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC NOTES
virtual_memory_kb	**bigint**	**Applies to:** SQL Server 2012 (11.x) through SQL Server 2017. Specifies the total amount of virtual address space available to the process in user mode. Not nullable.
bpool_commited	**int**	**Applies to:** SQL Server 2008 through SQL Server 2008 R2. Represents the committed memory in kilobytes (KB) in the memory manager. Does not include reserved memory in the memory manager. Not nullable.
committed_kb	**int**	**Applies to:** SQL Server 2012 (11.x) through SQL Server 2017. Represents the committed memory in kilobytes (KB) in the memory manager. Does not include reserved memory in the memory manager. Not nullable.
bpool_commit_target	**int**	**Applies to:** SQL Server 2008 through SQL Server 2008 R2. Represents the amount of memory, in kilobytes (KB), that can be consumed by SQL Server memory manager.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC NOTES
committed_target_kb	**int**	**Applies to:** SQL Server 2012 (11.x) through SQL Server 2017. Represents the amount of memory, in kilobytes (KB), that can be consumed by SQL Server memory manager. The target amount is calculated using a variety of inputs like: - the current state of the system including its load - the memory requested by current processes - the amount of memory installed on the computer - configuration parameters If **committed_target_kb** is larger than **committed_kb**, the memory manager will try to obtain additional memory. If **committed_target_kb** is smaller than **committed_kb**, the memory manager will try to shrink the amount of memory committed. The **committed_target_kb** always includes stolen and reserved memory. Not nullable.
bpool_visible	**int**	**Applies to:** SQL Server 2008 through SQL Server 2008 R2. Number of 8-KB buffers in the buffer pool that are directly accessible in the process virtual address space. When not using the Address Windowing Extensions (AWE), when the buffer pool has obtained its memory target (bpool_committed = bpool_commit_target), the value of bpool_visible equals the value of bpool_committed.When using AWE on a 32-bit version of SQL Server, bpool_visible represents the size of the AWE mapping window used to access physical memory allocated by the buffer pool. The size of this mapping window is bound by the process address space and, therefore, the visible amount will be smaller than the committed amount, and can be further reduced by internal components consuming memory for purposes other than database pages. If the value of bpool_visible is too low, you might receive out of memory errors.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC NOTES
visible_target_kb	**int**	**Applies to:** SQL Server 2012 (11.x) through SQL Server 2017. Is the same as **committed_target_kb**. Not nullable.
stack_size_in_bytes	**int**	Specifies the size of the call stack for each thread created by SQL Server. Not nullable.
os_quantum	**bigint**	Represents the Quantum for a non-preemptive task, measured in milliseconds. Quantum (in seconds) = **os_quantum** / CPU clock speed. Not nullable.
os_error_mode	**int**	Specifies the error mode for the SQL Server process. Not nullable.
os_priority_class	**int**	Specifies the priority class for the SQL Server process. Nullable. 32 = Normal (Error log will say SQL Server is starting at normal priority base (=7).) 128 = High (Error log will say SQL Server is running at high priority base. (=13).) For more information, see Configure the priority boost Server Configuration Option.
max_workers_count	**int**	Represents the maximum number of workers that can be created. Not nullable.
scheduler_count	**int**	Represents the number of user schedulers configured in the SQL Server process. Not nullable.
scheduler_total_count	**int**	Represents the total number of schedulers in SQL Server. Not nullable.
deadlock_monitor_serial_number	**int**	Specifies the ID of the current deadlock monitor sequence. Not nullable.
sqlserver_start_time_ms_ticks	**bigint**	Represents the **ms_tick** number when SQL Server last started. Compare to the current ms_ticks column. Not nullable.
sqlserver_start_time	**datetime**	Specifies the date and time SQL Server last started. Not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC NOTES
affinity_type	**int**	**Applies to:** SQL Server 2008 R2 through SQL Server 2017. Specifies the type of server CPU process affinity currently in use. Not nullable. For more information, see ALTER SERVER CONFIGURATION (Transact-SQL). 1 = MANUAL 2 = AUTO
affinity_type_desc	**varchar(60)**	**Applies to:** SQL Server 2008 R2 through SQL Server 2017. Describes the **affinity_type** column. Not nullable. MANUAL = affinity has been set for at least one CPU. AUTO = SQL Server can freely move threads between CPUs.
process_kernel_time_ms	**bigint**	**Applies to:** SQL Server 2008 R2 through SQL Server 2017. Total time in milliseconds spent by all SQL Server threads in kernel mode. This value can be larger than a single processor clock because it includes the time for all processors on the server. Not nullable.
process_user_time_ms	**bigint**	**Applies to:** SQL Server 2008 R2 through SQL Server 2017. Total time in milliseconds spent by all SQL Server threads in user mode. This value can be larger than a single processor clock because it includes the time for all processors on the server. Not nullable.
time_source	**int**	**Applies to:** SQL Server 2008 R2 through SQL Server 2017. Indicates the API that SQL Server is using to retrieve wall clock time. Not nullable. 0 = QUERY_PERFORMANCE_COUNTER 1 = MULTIMEDIA_TIMER

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC NOTES
time_source_desc	**nvarchar(60)**	**Applies to:** SQL Server 2008 R2 through SQL Server 2017. Describes the **time_source** column. Not nullable. QUERY_PERFORMANCE_COUNTER = the QueryPerformanceCounter API retrieves wall clock time. MULTIMEDIA_TIMER = The multimedia timer API that retrieves wall clock time.
virtual_machine_type	**int**	**Applies to:** SQL Server 2008 R2 through SQL Server 2017. Indicates whether SQL Server is running in a virtualized environment. Not nullable. 0 = NONE 1 = HYPERVISOR 2 = OTHER
virtual_machine_type_desc	**nvarchar(60)**	**Applies to:** SQL Server 2008 R2 through SQL Server 2017. Describes the **virtual_machine_type** column. Not nullable. NONE = SQL Server is not running inside a virtual machine. HYPERVISOR = SQL Server is running inside a hypervisor, which implies a hardware-assisted virtualization. When the Hyper_V role is installed, the hypervisor hosts the OS, so an instance running on the host OS is running in the hypervisor. OTHER = SQL Server is running inside a virtual machine that does not employ hardware assistant such as Microsoft Virtual PC.
softnuma_configuration	**int**	**Applies to:** SQL Server 2016 (13.x) through SQL Server 2017. Specifies the way NUMA nodes are configured. Not nullable. 0 = OFF indicates hardware default 1 = Automated soft-NUMA 2 = Manual soft-NUMA via registry

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC NOTES
softnuma_configuration_desc	**nvarchar(60)**	**Applies to:** SQL Server 2016 (13.x) through SQL Server 2017. OFF = Soft-NUMA feature is OFF ON = SQL Server automatically determines the NUMA node sizes for Soft-NUMA MANUAL = Manually configured soft-NUMA
process_physical_affinity	**nvarchar(3072)**	**Applies to:** Starting with SQL Server 2017 (14.x). Information yet to come.
sql_memory_model	**int**	**Applies to:** SQL Server 2012 (11.x) SP4, SQL Server 2016 (13.x) SP1 through SQL Server 2017. Specifies the memory model used by SQL Server to allocate memory. Not nullable. 1 = Conventional Memory Model 2 = Lock Pages in Memory 3 = Large Pages in Memory

COLUMN NAME	DATA TYPE	DESCRIPTION AND VERSION-SPECIFIC NOTES
sql_memory_model_desc	**nvarchar(120)**	**Applies to:** SQL Server 2012 (11.x) SP4, SQL Server 2016 (13.x) SP1 through SQL Server 2017. Specifies the memory model used by SQL Server to allocate memory. Not nullable. **CONVENTIONAL** = SQL Server is using Conventional Memory model to allocate memory. This is default sql memory model when SQL Server service account does not have Lock Pages in Memory privileges during startup. **LOCK_PAGES** = SQL Server is using Lock Pages in Memory to allocate memory. This is the default sql memory manager when SQL Server service account possess Lock Pages in Memory privilege during SQL Server startup. **LARGE_PAGES** = SQL Server is using Large Pages in Memory to allocate memory. SQL Server uses Large Pages allocator to allocate memory only with Enterprise edition when SQL Server service account possess Lock Pages in Memory privilege during server startup and when Trace Flag 834 is turned ON.
pdw_node_id	**int**	**Applies to:** Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.
socket_count	**int**	**Applies to:** SQL Server 2016 (13.x) SP2 through SQL Server 2017. Specifies the number of processor sockets available on the system.
cores_per_socket	**int**	**Applies to:** SQL Server 2016 (13.x) SP2 through SQL Server 2017. Specifies the number of processors per socket available on the system.
numa_node_count	**int**	**Applies to:** SQL Server 2016 (13.x) SP2 through SQL Server 2017. Specifies the number of numa nodes available on the system. This column includes physical numa nodes as well as soft numa nodes.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

Dynamic Management Views and Functions (Transact-SQL)
SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_sys_memory (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns memory information from the operating system.

SQL Server is bounded by, and responds to, external memory conditions at the operating system level and the physical limits of the underlying hardware. Determining the overall system state is an important part of evaluating SQL Server memory usage.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_sys_memory**.

COLUMN NAME	DATA TYPE	DESCRIPTION
total_physical_memory_kb	**bigint**	Total size of physical memory available to the operating system, in kilobytes (KB).
available_physical_memory_kb	**bigint**	Size of physical memory available, in KB.
total_page_file_kb	**bigint**	Size of the commit limit reported by the operating system in KB
available_page_file_kb	**bigint**	Total amount of page file thatis not being used, in KB.
system_cache_kb	**bigint**	Total amount of system cache memory, in KB.
kernel_paged_pool_kb	**bigint**	Total amount of the paged kernel pool, in KB.
kernel_nonpaged_pool_kb	**bigint**	Total amount of the nonpaged kernel pool, in KB.
system_high_memory_signal_state	**bit**	State of the system high memory resource notification. A value of 1 indicates the high memory signal has been set by Windows. For more information, see CreateMemoryResourceNotification in the MSDN library.

COLUMN NAME	DATA TYPE	DESCRIPTION
system_low_memory_signal_state	**bit**	State of the system low memory resource notification. A value of 1 indicates the low memory signal has been set by Windows. For more information, see CreateMemoryResourceNotification in the MSDN library.
system_memory_state_desc	**nvarchar(256)**	Description of the memory state. See the table below.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

CONDITION	VALUE
system_high_memory_signal_state = 1 and system_low_memory_signal_state = 0	Available physical memory is high
system_high_memory_signal_state = 0 and system_low_memory_signal_state = 1	Available physical memory is low
system_high_memory_signal_state = 0 and system_low_memory_signal_state = 0	Physical memory usage is steady
system_high_memory_signal_state = 1 and system_low_memory_signal_state = 1	Physical memory state is transitioning The high and low signal should never be on at the same time. However, rapid changes at the operating system level can cause both values to appear to be on to a user mode application. The appearance of both signals being on will be interpreted as a transition state.

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Dynamic Management Views and Functions (Transact-SQL)
SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_tasks (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row for each task that is active in the instance of SQL Server.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_tasks**.

COLUMN NAME	DATA TYPE	DESCRIPTION
task_address	**varbinary(8)**	Memory address of the object.
task_state	**nvarchar(60)**	State of the task. This can be one of the following: PENDING: Waiting for a worker thread. RUNNABLE: Runnable, but waiting to receive a quantum. RUNNING: Currently running on the scheduler. SUSPENDED: Has a worker, but is waiting for an event. DONE: Completed. SPINLOOP: Stuck in a spinlock.
context_switches_count	**int**	Number of scheduler context switches that this task has completed.
pending_io_count	**int**	Number of physical I/Os that are performed by this task.
pending_io_byte_count	**bigint**	Total byte count of I/Os that are performed by this task.
pending_io_byte_average	**int**	Average byte count of I/Os that are performed by this task.
scheduler_id	**int**	ID of the parent scheduler. This is a handle to the scheduler information for this task. For more information, see sys.dm_os_schedulers (Transact-SQL).
session_id	**smallint**	ID of the session that is associated with the task.

COLUMN NAME	DATA TYPE	DESCRIPTION
exec_context_id	**int**	Execution context ID that is associated with the task.
request_id	**int**	ID of the request of the task. For more information, see sys.dm_exec_requests (Transact-SQL).
worker_address	**varbinary(8)**	Memory address of the worker that is running the task. NULL = Task is either waiting for a worker to be able to run, or the task has just finished running. For more information, see sys.dm_os_workers (Transact-SQL).
host_address	**varbinary(8)**	Memory address of the host. 0 = Hosting was not used to create the task. This helps identify the host that was used to create this task. For more information, see sys.dm_os_hosts (Transact-SQL).
parent_task_address	**varbinary(8)**	Memory address of the task that is the parent of the object.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

A. Monitoring parallel requests

For requests that are executed in parallel, you will see multiple rows for the same combination of (<**session_id**>, <**request_id**>). Use the following query to find the Configure the max degree of parallelism Server Configuration Option for all active requests.

> **NOTE**
>
> A **request_id** is unique within a session.

```
SELECT
    task_address,
    task_state,
    context_switches_count,
    pending_io_count,
    pending_io_byte_count,
    pending_io_byte_average,
    scheduler_id,
    session_id,
    exec_context_id,
    request_id,
    worker_address,
    host_address
  FROM sys.dm_os_tasks
  ORDER BY session_id, request_id;
```

B. Associating session IDs with Windows threads

You can use the following query to associate a session ID value with a Windows thread ID. You can then monitor the performance of the thread in the Windows Performance Monitor. The following query does not return information for sessions that are sleeping.

```
SELECT STasks.session_id, SThreads.os_thread_id
  FROM sys.dm_os_tasks AS STasks
  INNER JOIN sys.dm_os_threads AS SThreads
    ON STasks.worker_address = SThreads.worker_address
  WHERE STasks.session_id IS NOT NULL
  ORDER BY STasks.session_id;
GO
```

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_threads (Transact-SQL)

5/4/2018 • 3 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a list of all SQL Server Operating System threads that are running under the SQL Server process.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_threads**.

COLUMN NAME	DATA TYPE	DESCRIPTION
thread_address	**varbinary(8)**	Memory address (Primary Key) of the thread.
started_by_sqlservr	**bit**	Indicates the thread initiator. 1 = SQL Server started the thread. 0 = Another component started the thread, such as an extended stored procedure from within SQL Server.
os_thread_id	**int**	ID of the thread that is assigned by the operating system.
status	**int**	Internal status flag.
instruction_address	**varbinary(8)**	Address of the instruction that is currently being executed.
creation_time	**datetime**	Time when this thread was created.
kernel_time	**bigint**	Amount of kernel time that is used by this thread.
usermode_time	**bigint**	Amount of user time that is used by this thread.
stack_base_address	**varbinary(8)**	Memory address of the highest stack address for this thread.
stack_end_address	**varbinary(8)**	Memory address of the lowest stack address of this thread.
stack_bytes_committed	**int**	Number of bytes that are committed in the stack.
stack_bytes_used	**int**	Number of bytes that are actively being used on the thread.

COLUMN NAME	DATA TYPE	DESCRIPTION
affinity	**bigint**	CPU mask on which this thread is running. This depends on the value configured by the **ALTER SERVER CONFIGURATION SET PROCESS AFFINITY** statement. Might be different from the scheduler in case of soft-affinity.
Priority	**int**	Priority value of this thread.
Locale	**int**	Cached locale LCID for the thread.
Token	**varbinary(8)**	Cached impersonation token handle for the thread.
is_impersonating	**int**	Indicates whether this thread is using Win32 impersonation. 1 = The thread is using security credentials that are different from the default of the process. This indicates that the thread is impersonating an entity other than the one that created the process.
is_waiting_on_loader_lock	**int**	Operating system status of whether the thread is waiting on the loader lock.
fiber_data	**varbinary(8)**	Current Win32 fiber that is running on the thread. This is only applicable when SQL Server is configured for lightweight pooling.
thread_handle	**varbinary(8)**	Internal use only.
event_handle	**varbinary(8)**	Internal use only.
scheduler_address	**varbinary(8)**	Memory address of the scheduler that is associated with this thread. For more information, see sys.dm_os_schedulers (Transact-SQL).
worker_address	**varbinary(8)**	Memory address of the worker that is bound to this thread. For more information, see sys.dm_os_workers (Transact-SQL).
fiber_context_address	**varbinary(8)**	Internal fiber context address. This is only applicable when SQL Server is configured for lightweight pooling.
self_address	**varbinary(8)**	Internal consistency pointer.

COLUMN NAME	DATA TYPE	DESCRIPTION
processor_group	**smallint**	**Applies to**: SQL Server 2008 R2 through SQL Server 2017. Processor group ID.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

Upon startup, SQL Server starts threads and then associates workers with those threads. However, external components, such as an extended stored procedure, can start threads under the SQL Server process. SQL Server has no control of these threads. sys.dm_os_threads can provide information about rogue threads that consume resources in the SQL Server process.

The following query is used to find workers, along with time used for execution, that are running threads not started by SQL Server.

> **NOTE**
> For conciseness, the following query uses an asterisk (`*`) in the `SELECT` statement. You should avoid using the asterisk (*), especially against catalog views, dynamic management views, and system table-valued functions. Future upgrades and releases of Microsoft SQL Server may add columns and change the order of columns to these views and functions. These changes might break applications that expect a particular order and number of columns.

```
SELECT *
  FROM sys.dm_os_threads
  WHERE started_by_sqlservr = 0;
```

See Also

[sys.dm_os_workers (Transact-SQL)]
[SQL Server Operating System Related Dynamic Management Views (Transact-SQL)]

sys.dm_os_virtual_address_dump (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about a range of pages in the virtual address space of the calling process.

> **NOTE**
>
> This information is also returned by the **VirtualQuery** Windows API.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_virtual_address_dump**.

COLUMN NAME	DATA TYPE	DESCRIPTION
region_base_address	**varbinary(8)**	Pointer to the base address of the region of pages. Is not nullable.
region_allocation_base_address	**varbinary(8)**	Pointer to the base address of a range of pages allocated by the VirtualAlloc Windows API function. The page pointed to by the BaseAddress member is contained within this allocation range. Is not nullable.
region_allocation_protection	**varbinary(8)**	Protection attributes when the region was first allocated. The value is one of the following: - PAGE_READONLY - PAGE_READWRITE - PAGE_NOACCESS - PAGE_WRITECOPY - PAGE_EXECUTE - PAGE_EXECUTE_READ - PAGE_EXECUTE_READWRITE - PAGE_EXECUTE_WRITECOPY - PAGE_GUARD - PAGE_NOCACHE Is not nullable.
region_size_in_bytes	**bigint**	Size of the region, in bytes, starting at the base address in which all the pages have the same attributes. Is not nullable.

COLUMN NAME	DATA TYPE	DESCRIPTION
region_state	**varbinary(8)**	Current state of the region. This is one of the following: - MEM_COMMIT - MEM_RESERVE - MEM_FREE Is not nullable.
region_current_protection	**varbinary(8)**	Protection attributes. The value is one of the following: - PAGE_READONLY - PAGE_READWRITE - PAGE_NOACCESS - PAGE_WRITECOPY - PAGE_EXECUTE - PAGE_EXECUTE_READ - PAGE_EXECUTE_READWRITE - PAGE_EXECUTE_WRITECOPY - PAGE_GUARD - PAGE_NOCACHE Is not nullable.
region_type	**varbinary(8)**	Identifies the types of pages in the region. The value can be one of the following: - MEM_PRIVATE - MEM_MAPPED - MEM_IMAGE Is not nullable.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

Requires VIEW SERVER STATE permission on the server.

See Also

Dynamic Management Views and Functions (Transact-SQL)
SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_volume_stats (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the operating system volume (directory) on which the specified databases and files are stored in SQL Server. Use this dynamic management function to check the attributes of the physical disk drive or return available free space information about the directory.

Transact-SQL Syntax Conventions

Syntax

```
sys.dm_os_volume_stats (database_id, file_id)
```

Arguments

database_id
ID of the database. *database_id* is **int**, with no default. Cannot be NULL.

file_id
ID of the file. *file_id* is **int**, with no default. Cannot be NULL.

Table Returned

Column	**Data type**	**Description**
database_id	**int**	ID of the database. Cannot be null.
file_id	**int**	ID of the file. Cannot be null.
volume_mount_point	**nvarchar(512)**	Mount point at which the volume is rooted. Can return an empty string.
volume_id	**nvarchar(512)**	Operating system volume ID. Can return an empty string
logical_volume_name	**nvarchar(512)**	Logical volume name. Can return an empty string
file_system_type	**nvarchar(512)**	Type of file system volume (for example, NTFS, FAT, RAW). Can return an empty string
total_bytes	**bigint**	Total size in bytes of the volume. Cannot be null.

available_bytes	**bigint**	Available free space on the volume. Cannot be null.
supports_compression	**bit**	Indicates if the volume supports operating system compression. Cannot be null.
supports_alternate_streams	**bit**	Indicates if the volume supports alternate streams. Cannot be null.
supports_sparse_files	**bit**	Indicates if the volume supports sparse files. Cannot be null.
is_read_only	**bit**	Indicates if the volume is currently marked as read only. Cannot be null.
is_compressed	**bit**	Indicates if this volume is currently compressed. Cannot be null.

Security

Permissions

Requires VIEW SERVER STATE permission.

Examples

A. Return total space and available space for all database files

The following example returns the total space and available space (in bytes) for all database files in the instance of SQL Server.

```
SELECT f.database_id, f.file_id, volume_mount_point, total_bytes, available_bytes
FROM sys.master_files AS f
CROSS APPLY sys.dm_os_volume_stats(f.database_id, f.file_id);
```

B. Return total space and available space for the current database

The following example returns the total space and available space (in bytes) for the database files in the current database.

```
SELECT database_id, f.file_id, volume_mount_point, total_bytes, available_bytes
FROM sys.database_files AS f
CROSS APPLY sys.dm_os_volume_stats(DB_ID(f.name), f.file_id);
```

See Also

sys.master_files (Transact-SQL)
sys.database_files (Transact-SQL)

sys.dm_os_waiting_tasks (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about the wait queue of tasks that are waiting on some resource.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_waiting_tasks**.

COLUMN NAME	DATA TYPE	DESCRIPTION
waiting_task_address	**varbinary(8)**	Address of the waiting task.
session_id	**smallint**	ID of the session associated with the task.
exec_context_id	**int**	ID of the execution context associated with the task.
wait_duration_ms	**bigint**	Total wait time for this wait type, in milliseconds. This time is inclusive of **signal_wait_time**.
wait_type	**nvarchar(60)**	Name of the wait type.
resource_address	**varbinary(8)**	Address of the resource for which the task is waiting.
blocking_task_address	**varbinary(8)**	Task that is currently holding this resource
blocking_session_id	**smallint**	ID of the session that is blocking the request. If this column is NULL, the request is not blocked, or the session information of the blocking session is not available (or cannot be identified). -2 = The blocking resource is owned by an orphaned distributed transaction. -3 = The blocking resource is owned by a deferred recovery transaction. -4 = Session ID of the blocking latch owner could not be determined due to internal latch state transitions.
blocking_exec_context_id	**int**	ID of the execution context of the blocking task.

COLUMN NAME	DATA TYPE	DESCRIPTION
resource_description	**nvarchar(3072)**	Description of the resource that is being consumed. For more information, see the list below.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

resource_description column

The resource_description column has the following possible values.

Thread-pool resource owner:

- threadpool id=scheduler<hex-address>

  Parallel query resource owner:

- exchangeEvent id={Port|Pipe}<hex-address> WaitType=<exchange-wait-type> nodeId=<exchange-node-id>

  Exchange-wait-type:

- e_waitNone
- e_waitPipeNewRow
- e_waitPipeGetRow
- e_waitSynchronizeConsumerOpen
- e_waitPortOpen
- e_waitPortClose
- e_waitRange

  Lock resource owner:

- <type-specific-description> id=lock<lock-hex-address> mode=<mode> associatedObjectId=<associated-obj-id>

  <type-specific-description> can be:

  - For DATABASE: databaselock subresource=<databaselock-subresource> dbid=<db-id>
  - For FILE: filelock fileid=<file-id> subresource=<filelock-subresource> dbid=<db-id>
  - For OBJECT: objectlock lockPartition=<lock-partition-id> objid=<obj-id> subresource=<objectlock-subresource> dbid=<db-id>
  - For PAGE: pagelock fileid=<file-id> pageid=<page-id> dbid=<db-id> subresource=<pagelock-subresource>
  - For Key: keylock hobtid=<hobt-id> dbid=<db-id>
  - For EXTENT: extentlock fileid=<file-id> pageid=<page-id> dbid=<db-id>

- For RID: ridlock fileid=<file-id> pageid=<page-id> dbid=<db-id>
- For APPLICATION: applicationlock hash=<hash> databasePrincipalId=<role-id> dbid=<db-id>
- For METADATA: metadatalock subresource=<metadata-subresource> classid=<metadatalock-description> dbid=<db-id>
- For HOBT: hobtlock hobtid=<hobt-id> subresource=<hobt-subresource> dbid=<db-id>
- For ALLOCATION_UNIT: allocunitlock hobtid=<hobt-id> subresource=<alloc-unit-subresource> dbid=<db-id>

  <mode> can be:

  Sch-S, Sch-M, S, U, X, IS, IU, IX, SIU, SIX, UIX, BU, RangeS-S, RangeS-U, RangeI-N, RangeI-S, RangeI-U, RangeI-X, RangeX-, RangeX-U, RangeX-X

External resource owner:

- External ExternalResource=<wait-type>

Generic resource owner:

- TransactionMutex TransactionInfo Workspace=<workspace-id>
- Mutex
- CLRTaskJoin
- CLRMonitorEvent
- CLRRWLockEvent
- resourceWait

Latch resource owner:

- <db-id>:<file-id>:<page-in-file>
- <GUID>
- <latch-class> (<latch-address>)

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Example

This example will identify blocked sessions. Execute the Transact-SQL query in SQL Server Management Studio.

```
SELECT * FROM sys.dm_os_waiting_tasks
WHERE blocking_session_id IS NOT NULL;
```

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

sys.dm_os_wait_stats (Transact-SQL)

5/3/2018 • 79 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about all the waits encountered by threads that executed. You can use this aggregated view to diagnose performance issues with SQL Server and also with specific queries and batches. sys.dm_exec_session_wait_stats (Transact-SQL) provides similar information by session.

> **NOTE**
> To call this from **Azure SQL Data Warehouse or Parallel Data Warehouse**, use the name **sys.dm_pdw_nodes_os_wait_stats**.

COLUMN NAME	DATA TYPE	DESCRIPTION
wait_type	**nvarchar(60)**	Name of the wait type. For more information, see Types of Waits, later in this topic.
waiting_tasks_count	**bigint**	Number of waits on this wait type. This counter is incremented at the start of each wait.
wait_time_ms	**bigint**	Total wait time for this wait type in milliseconds. This time is inclusive of signal_wait_time_ms.
max_wait_time_ms	**bigint**	Maximum wait time on this wait type.
signal_wait_time_ms	**bigint**	Difference between the time that the waiting thread was signaled and when it started running.
pdw_node_id	**int**	The identifier for the node that this distribution is on. **Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Types of Waits

Resource waits Resource waits occur when a worker requests access to a resource that is not available because the resource is being used by some other worker or is not yet available. Examples of resource waits are locks, latches, network and disk I/O waits. Lock and latch waits are waits on synchronization objects

Queue waits

Queue waits occur when a worker is idle, waiting for work to be assigned. Queue waits are most typically seen with system background tasks such as the deadlock monitor and deleted record cleanup tasks. These tasks will wait for work requests to be placed into a work queue. Queue waits may also periodically become active even if no new packets have been put on the queue.

External waits
External waits occur when a SQL Server worker is waiting for an external event, such as an extended stored procedure call or a linked server query, to finish. When you diagnose blocking issues, remember that external waits do not always imply that the worker is idle, because the worker may actively be running some external code.

`sys.dm_os_wait_stats` shows the time for waits that have completed. This dynamic management view does not show current waits.

A SQL Server worker thread is not considered to be waiting if any of the following is true:

- A resource becomes available.
- A queue is nonempty.
- An external process finishes.

  Although the thread is no longer waiting, the thread does not have to start running immediately. This is because such a thread is first put on the queue of runnable workers and must wait for a quantum to run on the scheduler.

  In SQL Server the wait-time counters are **bigint** values and therefore are not as prone to counter rollover as the equivalent counters in earlier versions of SQL Server.

  Specific types of wait times during query execution can indicate bottlenecks or stall points within the query. Similarly, high wait times, or wait counts server wide can indicate bottlenecks or hot spots in interaction query interactions within the server instance. For example, lock waits indicate data contention by queries; page IO latch waits indicate slow IO response times; page latch update waits indicate incorrect file layout.

  The contents of this dynamic management view can be reset by running the following command:

```
DBCC SQLPERF ('sys.dm_os_wait_stats', CLEAR);
GO
```

This command resets all counters to 0.

> **NOTE**
> These statistics are not persisted across SQL Server restarts, and all data is cumulative since the last time the statistics were reset or the server was started.

The following table lists the wait types encountered by tasks.

TYPE	DESCRIPTION
ABR	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

TYPE	DESCRIPTION
AM_INDBUILD_ALLOCATION	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
AM_SCHEMAMGR_UNSHARED_CACHE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
ASSEMBLY_FILTER_HASHTABLE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
ASSEMBLY_LOAD	Occurs during exclusive access to assembly loading.
ASYNC_DISKPOOL_LOCK	Occurs when there is an attempt to synchronize parallel threads that are performing tasks such as creating or initializing a file.
ASYNC_IO_COMPLETION	Occurs when a task is waiting for I/Os to finish.
ASYNC_NETWORK_IO	Occurs on network writes when the task is blocked behind the network. Verify that the client is processing data from the server.
ASYNC_OP_COMPLETION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
ASYNC_OP_CONTEXT_READ	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
ASYNC_OP_CONTEXT_WRITE	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
ASYNC_SOCKETDUP_IO	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
AUDIT_GROUPCACHE_LOCK	Occurs when there is a wait on a lock that controls access to a special cache. The cache contains information about which audits are being used to audit each audit action group.
AUDIT_LOGINCACHE_LOCK	Occurs when there is a wait on a lock that controls access to a special cache. The cache contains information about which audits are being used to audit login audit action groups.
AUDIT_ON_DEMAND_TARGET_LOCK	Occurs when there is a wait on a lock that is used to ensure single initialization of audit related Extended Event targets.
AUDIT_XE_SESSION_MGR	Occurs when there is a wait on a lock that is used to synchronize the starting and stopping of audit related Extended Events sessions.

TYPE	DESCRIPTION
BACKUP	Occurs when a task is blocked as part of backup processing.
BACKUP_OPERATOR	Occurs when a task is waiting for a tape mount. To view the tape status, query sys.dm_io_backup_tapes. If a mount operation is not pending, this wait type may indicate a hardware problem with the tape drive.
BACKUPBUFFER	Occurs when a backup task is waiting for data, or is waiting for a buffer in which to store data. This type is not typical, except when a task is waiting for a tape mount.
BACKUPIO	Occurs when a backup task is waiting for data, or is waiting for a buffer in which to store data. This type is not typical, except when a task is waiting for a tape mount.
BACKUPTHREAD	Occurs when a task is waiting for a backup task to finish. Wait times may be long, from several minutes to several hours. If the task that is being waited on is in an I/O process, this type does not indicate a problem.
BAD_PAGE_PROCESS	Occurs when the background suspect page logger is trying to avoid running more than every five seconds. Excessive suspect pages cause the logger to run frequently.
BLOB_METADATA	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
BMPALLOCATION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
BMPBUILD	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
BMPREPARTITION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
BMPREPLICATION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
BPSORT	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
BROKER_CONNECTION_RECEIVE_TASK	Occurs when waiting for access to receive a message on a connection endpoint. Receive access to the endpoint is serialized.
BROKER_DISPATCHER	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
BROKER_ENDPOINT_STATE_MUTEX	Occurs when there is contention to access the state of a Service Broker connection endpoint. Access to the state for changes is serialized.
BROKER_EVENTHANDLER	Occurs when a task is waiting in the primary event handler of the Service Broker. This should occur very briefly.
BROKER_FORWARDER	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
BROKER_INIT	Occurs when initializing Service Broker in each active database. This should occur infrequently.
BROKER_MASTERSTART	Occurs when a task is waiting for the primary event handler of the Service Broker to start. This should occur very briefly.
BROKER_RECEIVE_WAITFOR	Occurs when the RECEIVE WAITFOR is waiting. This may mean that either no messages are ready to be received in the queue or a lock contention is preventing it from receiving messages from the queue.
BROKER_REGISTERALLENDPOINTS	Occurs during the initialization of a Service Broker connection endpoint. This should occur very briefly.
BROKER_SERVICE	Occurs when the Service Broker destination list that is associated with a target service is updated or re-prioritized.
BROKER_SHUTDOWN	Occurs when there is a planned shutdown of Service Broker. This should occur very briefly, if at all.
BROKER_START	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
BROKER_TASK_SHUTDOWN	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
BROKER_TASK_STOP	Occurs when the Service Broker queue task handler tries to shut down the task. The state check is serialized and must be in a running state beforehand.
BROKER_TASK_SUBMIT	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
BROKER_TO_FLUSH	Occurs when the Service Broker lazy flusher flushes the in-memory transmission objects to a work table.
BROKER_TRANSMISSION_OBJECT	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
BROKER_TRANSMISSION_TABLE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
BROKER_TRANSMISSION_WORK	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
BROKER_TRANSMITTER	Occurs when the Service Broker transmitter is waiting for work.
BUILTIN_HASHKEY_MUTEX	May occur after startup of instance, while internal data structures are initializing. Will not recur once data structures have initialized.
CHANGE_TRACKING_WAITFORCHANGES	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
CHECK_PRINT_RECORD	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
CHECK_SCANNER_MUTEX	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
CHECK_TABLES_INITIALIZATION	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
CHECK_TABLES_SINGLE_SCAN	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
CHECK_TABLES_THREAD_BARRIER	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
CHECKPOINT_QUEUE	Occurs while the checkpoint task is waiting for the next checkpoint request.
CHKPT	Occurs at server startup to tell the checkpoint thread that it can start.
CLEAR_DB	Occurs during operations that change the state of a database, such as opening or closing a database.
CLR_AUTO_EVENT	Occurs when a task is currently performing common language runtime (CLR) execution and is waiting for a particular autoevent to be initiated. Long waits are typical, and do not indicate a problem.
CLR_CRST	Occurs when a task is currently performing CLR execution and is waiting to enter a critical section of the task that is currently being used by another task.

TYPE	DESCRIPTION
CLR_JOIN	Occurs when a task is currently performing CLR execution and waiting for another task to end. This wait state occurs when there is a join between tasks.
CLR_MANUAL_EVENT	Occurs when a task is currently performing CLR execution and is waiting for a specific manual event to be initiated.
CLR_MEMORY_SPY	Occurs during a wait on lock acquisition for a data structure that is used to record all virtual memory allocations that come from CLR. The data structure is locked to maintain its integrity if there is parallel access.
CLR_MONITOR	Occurs when a task is currently performing CLR execution and is waiting to obtain a lock on the monitor.
CLR_RWLOCK_READER	Occurs when a task is currently performing CLR execution and is waiting for a reader lock.
CLR_RWLOCK_WRITER	Occurs when a task is currently performing CLR execution and is waiting for a writer lock.
CLR_SEMAPHORE	Occurs when a task is currently performing CLR execution and is waiting for a semaphore.
CLR_TASK_START	Occurs while waiting for a CLR task to complete startup.
CLRHOST_STATE_ACCESS	Occurs where there is a wait to acquire exclusive access to the CLR-hosting data structures. This wait type occurs while setting up or tearing down the CLR runtime.
CMEMPARTITIONED	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
CMEMTHREAD	Occurs when a task is waiting on a thread-safe memory object. The wait time might increase when there is contention caused by multiple tasks trying to allocate memory from the same memory object.
COLUMNSTORE_BUILD_THROTTLE	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
COLUMNSTORE_COLUMNDATASET_SESSION_LIST	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
COMMIT_TABLE	TBD
CONNECTION_ENDPOINT_LOCK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

TYPE	DESCRIPTION
COUNTRECOVERYMGR	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
CREATE_DATINISERVICE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
CXCONSUMER	Occurs with parallel query plans when a consumer thread waits for a producer thread to send rows. This is a normal part of parallel query execution. **Applies to**: SQL Server (Starting with SQL Server 2016 (13.x) SP2, SQL Server 2017 (14.x) CU3), SQL Database
CXPACKET	Occurs with parallel query plans when synchronizing the query processor exchange iterator, and when producing and consuming rows. If waiting is excessive and cannot be reduced by tuning the query (such as adding indexes), consider adjusting the cost threshold for parallelism or lowering the degree of parallelism. **Note:** Starting with SQL Server 2016 (13.x) SP2, SQL Server 2017 (14.x) CU3, and SQL Database, CXPACKET only refers to synchronizing the query processor exchange iterator, and to producing rows for consumer threads. Consumer threads are tracked separately in the CXCONSUMER wait type.
CXROWSET_SYNC	Occurs during a parallel range scan.
DAC_INIT	Occurs while the dedicated administrator connection is initializing.
DBCC_SCALE_OUT_EXPR_CACHE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
DBMIRROR_DBM_EVENT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
DBMIRROR_DBM_MUTEX	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
DBMIRROR_EVENTS_QUEUE	Occurs when database mirroring waits for events to process.
DBMIRROR_SEND	Occurs when a task is waiting for a communications backlog at the network layer to clear to be able to send messages. Indicates that the communications layer is starting to become overloaded and affect the database mirroring data throughput.
DBMIRROR_WORKER_QUEUE	Indicates that the database mirroring worker task is waiting for more work.
DBMIRRORING_CMD	Occurs when a task is waiting for log records to be flushed to disk. This wait state is expected to be held for long periods of time.

TYPE	DESCRIPTION
DBSEEDING_FLOWCONTROL	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
DBSEEDING_OPERATION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
DEADLOCK_ENUM_MUTEX	Occurs when the deadlock monitor and sys.dm_os_waiting_tasks try to make sure that SQL Server is not running multiple deadlock searches at the same time.
DEADLOCK_TASK_SEARCH	Large waiting time on this resource indicates that the server is executing queries on top of sys.dm_os_waiting_tasks, and these queries are blocking deadlock monitor from running deadlock search. This wait type is used by deadlock monitor only. Queries on top of sys.dm_os_waiting_tasks use DEADLOCK_ENUM_MUTEX.
DEBUG	Occurs during Transact-SQL and CLR debugging for internal synchronization.
DIRECTLOGCONSUMER_LIST	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
DIRTY_PAGE_POLL	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
DIRTY_PAGE_SYNC	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
DIRTY_PAGE_TABLE_LOCK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
DISABLE_VERSIONING	Occurs when SQL Server polls the version transaction manager to see whether the timestamp of the earliest active transaction is later than the timestamp of when the state started changing. If this is this case, all the snapshot transactions that were started before the ALTER DATABASE statement was run have finished. This wait state is used when SQL Server disables versioning by using the ALTER DATABASE statement.
DISKIO_SUSPEND	Occurs when a task is waiting to access a file when an external backup is active. This is reported for each waiting user process. A count larger than five per user process may indicate that the external backup is taking too much time to finish.
DISPATCHER_PRIORITY_QUEUE_SEMAPHORE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
DISPATCHER_QUEUE_SEMAPHORE	Occurs when a thread from the dispatcher pool is waiting for more work to process. The wait time for this wait type is expected to increase when the dispatcher is idle.
DLL_LOADING_MUTEX	Occurs once while waiting for the XML parser DLL to load.
DPT_ENTRY_LOCK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
DROP_DATABASE_TIMER_TASK	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
DROPTEMP	Occurs between attempts to drop a temporary object if the previous attempt failed. The wait duration grows exponentially with each failed drop attempt.
DTC	Occurs when a task is waiting on an event that is used to manage state transition. This state controls when the recovery of Microsoft Distributed Transaction Coordinator (MS DTC) transactions occurs after SQL Server receives notification that the MS DTC service has become unavailable.
DTC_ABORT_REQUEST	Occurs in a MS DTC worker session when the session is waiting to take ownership of a MS DTC transaction. After MS DTC owns the transaction, the session can roll back the transaction. Generally, the session will wait for another session that is using the transaction.
DTC_RESOLVE	Occurs when a recovery task is waiting for the master database in a cross-database transaction so that the task can query the outcome of the transaction.
DTC_STATE	Occurs when a task is waiting on an event that protects changes to the internal MS DTC global state object. This state should be held for very short periods of time.
DTC_TMDOWN_REQUEST	Occurs in a MS DTC worker session when SQL Server receives notification that the MS DTC service is not available. First, the worker will wait for the MS DTC recovery process to start. Then, the worker waits to obtain the outcome of the distributed transaction that the worker is working on. This may continue until the connection with the MS DTC service has been reestablished.
DTC_WAITFOR_OUTCOME	Occurs when recovery tasks wait for MS DTC to become active to enable the resolution of prepared transactions.
DTCNEW_ENLIST	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
DTCNEW_PREPARE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

TYPE	DESCRIPTION
DTCNEW_RECOVERY	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
DTCNEW_TM	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
DTCNEW_TRANSACTION_ENLISTMENT	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
DTCPNTSYNC	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
DUMP_LOG_COORDINATOR	Occurs when a main task is waiting for a subtask to generate data. Ordinarily, this state does not occur. A long wait indicates an unexpected blockage. The subtask should be investigated.
DUMP_LOG_COORDINATOR_QUEUE	TBD
DUMPTRIGGER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
EC	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
EE_PMOLOCK	Occurs during synchronization of certain types of memory allocations during statement execution.
EE_SPECPROC_MAP_INIT	Occurs during synchronization of internal procedure hash table creation. This wait can only occur during the initial accessing of the hash table after the SQL Server instance starts.
ENABLE_EMPTY_VERSIONING	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
ENABLE_VERSIONING	Occurs when SQL Server waits for all update transactions in this database to finish before declaring the database ready to transition to snapshot isolation allowed state. This state is used when SQL Server enables snapshot isolation by using the ALTER DATABASE statement.
ERROR_REPORTING_MANAGER	Occurs during synchronization of multiple concurrent error log initializations.
EXCHANGE	Occurs during synchronization in the query processor exchange iterator during parallel queries.

TYPE	DESCRIPTION
EXECSYNC	Occurs during parallel queries while synchronizing in query processor in areas not related to the exchange iterator. Examples of such areas are bitmaps, large binary objects (LOBs), and the spool iterator. LOBs may frequently use this wait state.
EXECUTION_PIPE_EVENT_INTERNAL	Occurs during synchronization between producer and consumer parts of batch execution that are submitted through the connection context.
EXTERNAL_RG_UPDATE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
EXTERNAL_SCRIPT_NETWORK_IO	TBD **Applies to**: SQL Server 2017 (14.x) through current.
EXTERNAL_SCRIPT_PREPARE_SERVICE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
EXTERNAL_SCRIPT_SHUTDOWN	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
EXTERNAL_WAIT_ON_LAUNCHER,	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
FABRIC_HADR_TRANSPORT_CONNECTION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
FABRIC_REPLICA_CONTROLLER_LIST	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
FABRIC_REPLICA_CONTROLLER_STATE_AND_CONFIG	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
FABRIC_REPLICA_PUBLISHER_EVENT_PUBLISH	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
FABRIC_REPLICA_PUBLISHER_SUBSCRIBER_LIST	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
FABRIC_WAIT_FOR_BUILD_REPLICA_EVENT_PROCESSING	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
FAILPOINT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

TYPE	DESCRIPTION
FCB_REPLICA_READ	Occurs when the reads of a snapshot (or a temporary snapshot created by DBCC) sparse file are synchronized.
FCB_REPLICA_WRITE	Occurs when the pushing or pulling of a page to a snapshot (or a temporary snapshot created by DBCC) sparse file is synchronized.
FEATURE_SWITCHES_UPDATE	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
FFT_NSO_DB_KILL_FLAG	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_NSO_DB_LIST	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_NSO_FCB	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_NSO_FCB_FIND	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_NSO_FCB_PARENT	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_NSO_FCB_RELEASE_CACHED_ENTRIES	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_NSO_FCB_STATE	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
FFT_NSO_FILEOBJECT	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_NSO_TABLE_LIST	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_NTFS_STORE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_RECOVERY	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
FFT_RSFX_COMM	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_RSFX_WAIT_FOR_MEMORY	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_STARTUP_SHUTDOWN	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_STORE_DB	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_STORE_ROWSET_LIST	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FFT_STORE_TABLE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FILE_VALIDATION_THREADS	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
FILESTREAM_CACHE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FILESTREAM_CHUNKER	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FILESTREAM_CHUNKER_INIT	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FILESTREAM_FCB	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FILESTREAM_FILE_OBJECT	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FILESTREAM_WORKITEM_QUEUE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FILETABLE_SHUTDOWN	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
FOREIGN_REDO	TBD **Applies to**: SQL Server 2017 (14.x) through current.
FORWARDER_TRANSITION	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
FS_FC_RWLOCK	Occurs when there is a wait by the FILESTREAM garbage collector to do either of the following:
FS_GARBAGE_COLLECTOR_SHUTDOWN	Occurs when the FILESTREAM garbage collector is waiting for cleanup tasks to be completed.
FS_HEADER_RWLOCK	Occurs when there is a wait to acquire access to the FILESTREAM header of a FILESTREAM data container to either read or update contents in the FILESTREAM header file (Filestream.hdr).
FS_LOGTRUNC_RWLOCK	Occurs when there is a wait to acquire access to FILESTREAM log truncation to do either of the following:
FSA_FORCE_OWN_XACT	Occurs when a FILESTREAM file I/O operation needs to bind to the associated transaction, but the transaction is currently owned by another session.
FSAGENT	Occurs when a FILESTREAM file I/O operation is waiting for a FILESTREAM agent resource that is being used by another file I/O operation.
FSTR_CONFIG_MUTEX	Occurs when there is a wait for another FILESTREAM feature reconfiguration to be completed.
FSTR_CONFIG_RWLOCK	Occurs when there is a wait to serialize access to the FILESTREAM configuration parameters.
FT_COMPROWSET_RWLOCK	Full-text is waiting on fragment metadata operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_IFTS_RWLOCK	Full-text is waiting on internal synchronization. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_IFTS_SCHEDULER_IDLE_WAIT	Full-text scheduler sleep wait type. The scheduler is idle.
FT_IFTSHC_MUTEX	Full-text is waiting on an fdhost control operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_IFTSISM_MUTEX	Full-text is waiting on communication operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.

TYPE	DESCRIPTION
FT_MASTER_MERGE	Full-text is waiting on master merge operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_MASTER_MERGE_COORDINATOR	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FT_METADATA_MUTEX	Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.
FT_PROPERTYLIST_CACHE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
FT_RESTART_CRAWL	Occurs when a full-text crawl needs to restart from a last known good point to recover from a transient failure. The wait lets the worker tasks currently working on that population to complete or exit the current step.
FULLTEXT GATHERER	Occurs during synchronization of full-text operations.
GDMA_GET_RESOURCE_OWNER	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
GHOSTCLEANUP_UPDATE_STATS	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
GHOSTCLEANUPSYNCMGR	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
GLOBAL_QUERY_CANCEL	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
GLOBAL_QUERY_CLOSE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
GLOBAL_QUERY_CONSUMER	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
GLOBAL_QUERY_PRODUCER	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
GLOBAL_TRAN_CREATE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

TYPE	DESCRIPTION
GLOBAL_TRAN_UCS_SESSION	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
GUARDIAN	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
HADR_AG_MUTEX	Occurs when an Always On DDL statement or Windows Server Failover Clustering command is waiting for exclusive read/write access to the configuration of an availability group., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_AR_CRITICAL_SECTION_ENTRY	Occurs when an Always On DDL statement or Windows Server Failover Clustering command is waiting for exclusive read/write access to the runtime state of the local replica of the associated availability group., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_AR_MANAGER_MUTEX	Occurs when an availability replica shutdown is waiting for startup to complete or an availability replica startup is waiting for shutdown to complete. Internal use only., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_AR_UNLOAD_COMPLETED	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_ARCONTROLLER_NOTIFICATIONS_SUBSCRIBER_LIST	The publisher for an availability replica event (such as a state change or configuration change) is waiting for exclusive read/write access to the list of event subscribers. Internal use only., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_BACKUP_BULK_LOCK	The Always On primary database received a backup request from a secondary database and is waiting for the background thread to finish processing the request on acquiring or releasing the BulkOp lock., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_BACKUP_QUEUE	The backup background thread of the Always On primary database is waiting for a new work request from the secondary database. (typically, this occurs when the primary database is holding the BulkOp log and is waiting for the secondary database to indicate that the primary database can release the lock)., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
HADR_CLUSAPI_CALL	A SQL Server thread is waiting to switch from non-preemptive mode (scheduled by SQL Server) to preemptive mode (scheduled by the operating system) in order to invoke Windows Server Failover Clustering APIs., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_COMPRESSED_CACHE_SYNC	Waiting for access to the cache of compressed log blocks that is used to avoid redundant compression of the log blocks sent to multiple secondary databases., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_CONNECTIVITY_INFO	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_DATABASE_FLOW_CONTROL	Waiting for messages to be sent to the partner when the maximum number of queued messages has been reached. Indicates that the log scans are running faster than the network sends. This is an issue only if network sends are slower than expected., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_DATABASE_VERSIONING_STATE	Occurs on the versioning state change of an Always On secondary database. This wait is for internal data structures and is usually is very short with no direct effect on data access., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_DATABASE_WAIT_FOR_RECOVERY	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HADR_DATABASE_WAIT_FOR_RESTART	Waiting for the database to restart under Always On Availability Groups control. Under normal conditions, this is not a customer issue because waits are expected here., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_DATABASE_WAIT_FOR_TRANSITION_TO_VERSIONING	A query on object(s) in a readable secondary database of an Always On availability group is blocked on row versioning while waiting for commit or rollback of all transactions that were in-flight when the secondary replica was enabled for read workloads. This wait type guarantees that row versions are available before execution of a query under snapshot isolation., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_DB_COMMAND	Waiting for responses to conversational messages (which require an explicit response from the other side, using the Always On conversational message infrastructure). A number of different message types use this wait type., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
HADR_DB_OP_COMPLETION_SYNC	Waiting for responses to conversational messages (which require an explicit response from the other side, using the Always On conversational message infrastructure). A number of different message types use this wait type., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_DB_OP_START_SYNC	An Always On DDL statement or a Windows Server Failover Clustering command is waiting for serialized access to an availability database and its runtime state., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_DBR_SUBSCRIBER	The publisher for an availability replica event (such as a state change or configuration change) is waiting for exclusive read/write access to the runtime state of an event subscriber that corresponds to an availability database. Internal use only., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_DBR_SUBSCRIBER_FILTER_LIST	The publisher for an availability replica event (such as a state change or configuration change) is waiting for exclusive read/write access to the list of event subscribers that correspond to availability databases. Internal use only., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_DBSEEDING	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
HADR_DBSEEDING_LIST	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
HADR_DBSTATECHANGE_SYNC	Concurrency control wait for updating the internal state of the database replica., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_FABRIC_CALLBACK	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
HADR_FILESTREAM_BLOCK_FLUSH	The FILESTREAM Always On transport manager is waiting until processing of a log block is finished., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_FILESTREAM_FILE_CLOSE	The FILESTREAM Always On transport manager is waiting until the next FILESTREAM file gets processed and its handle gets closed., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
HADR_FILESTREAM_FILE_REQUEST	An Always On secondary replica is waiting for the primary replica to send all requested FILESTREAM files during UNDO., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_FILESTREAM_IOMGR	The FILESTREAM Always On transport manager is waiting for R/W lock that protects the FILESTREAM Always On I/O manager during startup or shutdown., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_FILESTREAM_IOMGR_IOCOMPLETION	The FILESTREAM Always On I/O manager is waiting for I/O completion., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_FILESTREAM_MANAGER	The FILESTREAM Always On transport manager is waiting for the R/W lock that protects the FILESTREAM Always On transport manager during startup or shutdown., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_FILESTREAM_PREPROC	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HADR_GROUP_COMMIT	Transaction commit processing is waiting to allow a group commit so that multiple commit log records can be put into a single log block. This wait is an expected condition that optimizes the log I/O, capture, and send operations., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_LOGCAPTURE_SYNC	Concurrency control around the log capture or apply object when creating or destroying scans. This is an expected wait when partners change state or connection status., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_LOGCAPTURE_WAIT	Waiting for log records to become available. Can occur either when waiting for new log records to be generated by connections or for I/O completion when reading log not in the cache. This is an expected wait if the log scan is caught up to the end of log or is reading from disk., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_LOGPROGRESS_SYNC	Concurrency control wait when updating the log progress status of database replicas., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_NOTIFICATION_DEQUEUE	A background task that processes Windows Server Failover Clustering notifications is waiting for the next notification. Internal use only., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
HADR_NOTIFICATION_WORKER_EXCLUSIVE_ACCESS	The Always On availability replica manager is waiting for serialized access to the runtime state of a background task that processes Windows Server Failover Clustering notifications. Internal use only., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_NOTIFICATION_WORKER_STARTUP_SYNC	A background task is waiting for the completion of the startup of a background task that processes Windows Server Failover Clustering notifications. Internal use only., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_NOTIFICATION_WORKER_TERMINATION_SYNC	A background task is waiting for the termination of a background task that processes Windows Server Failover Clustering notifications. Internal use only., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_PARTNER_SYNC	Concurrency control wait on the partner list., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_READ_ALL_NETWORKS	Waiting to get read or write access to the list of WSFC networks. Internal use only. Note: The engine keeps a list of WSFC networks that is used in dynamic management views (such as sys.dm_hadr_cluster_networks) or to validate Always On Transact-SQL statements that reference WSFC network information. This list is updated upon engine startup, WSFC related notifications, and internal Always On restart (for example, losing and regaining of WSFC quorum). Tasks will usually be blocked when an update in that list is in progress. , **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_RECOVERY_WAIT_FOR_CONNECTION	Waiting for the secondary database to connect to the primary database before running recovery. This is an expected wait, which can lengthen if the connection to the primary is slow to establish., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_RECOVERY_WAIT_FOR_UNDO	Database recovery is waiting for the secondary database to finish the reverting and initializing phase to bring it back to the common log point with the primary database. This is an expected wait after failovers.Undo progress can be tracked through the Windows System Monitor (perfmon.exe) and dynamic management views., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_REPLICAINFO_SYNC	Waiting for concurrency control to update the current replica state., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
HADR_SEEDING_CANCELLATION	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HADR_SEEDING_FILE_LIST	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HADR_SEEDING_LIMIT_BACKUPS	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HADR_SEEDING_SYNC_COMPLETION	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HADR_SEEDING_TIMEOUT_TASK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HADR_SEEDING_WAIT_FOR_COMPLETION	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HADR_SYNC_COMMIT	Waiting for transaction commit processing for the synchronized secondary databases to harden the log. This wait is also reflected by the Transaction Delay performance counter. This wait type is expected for synchronized availability groups and indicates the time to send, write, and acknowledge log to the secondary databases., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_SYNCHRONIZING_THROTTLE	Waiting for transaction commit processing to allow a synchronizing secondary database to catch up to the primary end of log in order to transition to the synchronized state. This is an expected wait when a secondary database is catching up., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_TDS_LISTENER_SYNC	Either the internal Always On system or the WSFC cluster will request that listeners are started or stopped. The processing of this request is always asynchronous, and there is a mechanism to remove redundant requests. There are also moments that this process is suspended because of configuration changes. All waits related with this listener synchronization mechanism use this wait type. Internal use only., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
HADR_TDS_LISTENER_SYNC_PROCESSING	Used at the end of an Always On Transact-SQL statement that requires starting and/or stopping anavailability group listener. Since the start/stop operation is done asynchronously, the user thread will block using this wait type until the situation of the listener is known., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_THROTTLE_LOG_RATE_GOVERNOR	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HADR_THROTTLE_LOG_RATE_LOG_SIZE	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
HADR_THROTTLE_LOG_RATE_SEEDING	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
HADR_THROTTLE_LOG_RATE_SEND_RECV_QUEUE_SIZE	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
HADR_TIMER_TASK	Waiting to get the lock on the timer task object and is also used for the actual waits between times that work is being performed. For example, for a task that runs every 10 seconds, after one execution, Always On Availability Groups waits about 10 seconds to reschedule the task, and the wait is included here., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_TRANSPORT_DBRLIST	Waiting for access to the transport layer's database replica list. Used for the spinlock that grants access to it., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_TRANSPORT_FLOW_CONTROL	Waiting when the number of outstanding unacknowledged Always On messages is over the out flow control threshold. This is on an availability replica-to-replica basis (not on a database-to-database basis)., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_TRANSPORT_SESSION	Always On Availability Groups is waiting while changing or accessing the underlying transport state., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_WORK_POOL	Concurrency control wait on the Always On Availability Groups background work task object., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
HADR_WORK_QUEUE	Always On Availability Groups background worker thread waiting for new work to be assigned. This is an expected wait when there are ready workers waiting for new work, which is the normal state., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HADR_XRF_STACK_ACCESS	Accessing (look up, add, and delete) the extended recovery fork stack for an Always On availability database., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HCCO_CACHE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HK_RESTORE_FILEMAP	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HKCS_PARALLEL_MIGRATION	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HKCS_PARALLEL_RECOVERY	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
HTBUILD	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HTDELETE	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
HTMEMO	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
HTREINIT	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
HTREPARTITION	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
HTTP_ENUMERATION	Occurs at startup to enumerate the HTTP endpoints to start HTTP.
HTTP_START	Occurs when a connection is waiting for HTTP to complete initialization.

TYPE	DESCRIPTION
HTTP_STORAGE_CONNECTION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
IMPPROV_IOWAIT	Occurs when SQL Server waits for a bulkload I/O to finish.
INSTANCE_LOG_RATE_GOVERNOR	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
INTERNAL_TESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
IO_AUDIT_MUTEX	Occurs during synchronization of trace event buffers.
IO_COMPLETION	Occurs while waiting for I/O operations to complete. This wait type generally represents non-data page I/Os. Data page I/O completion waits appear as PAGEIOLATCH_* waits.
IO_QUEUE_LIMIT	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
IO_RETRY	Occurs when an I/O operation such as a read or a write to disk fails because of insufficient resources, and is then retried.
IOAFF_RANGE_QUEUE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
KSOURCE_WAKEUP	Used by the service control task while waiting for requests from the Service Control Manager. Long waits are expected and do not indicate a problem.
KTM_ENLISTMENT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
KTM_RECOVERY_MANAGER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
KTM_RECOVERY_RESOLUTION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
LATCH_DT	Occurs when waiting for a DT (destroy) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.
LATCH_EX	Occurs when waiting for an EX (exclusive) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.

TYPE	DESCRIPTION
LATCH_KP	Occurs when waiting for a KP (keep) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.
LATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
LATCH_SH	Occurs when waiting for an SH (share) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.
LATCH_UP	Occurs when waiting for an UP (update) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH_* waits is available in sys.dm_os_latch_stats. Note that sys.dm_os_latch_stats groups LATCH_NL, LATCH_SH, LATCH_UP, LATCH_EX, and LATCH_DT waits together.
LAZYWRITER_SLEEP	Occurs when lazywriter tasks are suspended. This is a measure of the time spent by background tasks that are waiting. Do not consider this state when you are looking for user stalls.
LCK_M_BU	Occurs when a task is waiting to acquire a Bulk Update (BU) lock.
LCK_M_BU_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Bulk Update (BU) lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_BU_LOW_PRIORITY	Occurs when a task is waiting to acquire a Bulk Update (BU) lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_IS	Occurs when a task is waiting to acquire an Intent Shared (IS) lock.
LCK_M_IS_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Intent Shared (IS) lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_IS_LOW_PRIORITY	Occurs when a task is waiting to acquire an Intent Shared (IS) lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.

TYPE	DESCRIPTION
LCK_M_IU	Occurs when a task is waiting to acquire an Intent Update (IU) lock.
LCK_M_IU_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Intent Update (IU) lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_IU_LOW_PRIORITY	Occurs when a task is waiting to acquire an Intent Update (IU) lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_IX	Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock.
LCK_M_IX_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_IX_LOW_PRIORITY	Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RIn_NL	Occurs when a task is waiting to acquire a NULL lock on the current key value, and an Insert Range lock between the current and previous key. A NULL lock on the key is an instant release lock.
LCK_M_RIn_NL_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a NULL lock with Abort Blockers on the current key value, and an Insert Range lock with Abort Blockers between the current and previous key. A NULL lock on the key is an instant release lock. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RIn_NL_LOW_PRIORITY	Occurs when a task is waiting to acquire a NULL lock with Low Priority on the current key value, and an Insert Range lock with Low Priority between the current and previous key. A NULL lock on the key is an instant release lock. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RIn_S	Occurs when a task is waiting to acquire a shared lock on the current key value, and an Insert Range lock between the current and previous key.

TYPE	DESCRIPTION
LCK_M_RIn_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a shared lock with Abort Blockers on the current key value, and an Insert Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RIn_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a shared lock with Low Priority on the current key value, and an Insert Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RIn_U	Task is waiting to acquire an Update lock on the current key value, and an Insert Range lock between the current and previous key.
LCK_M_RIn_U_ABORT_BLOCKERS	Task is waiting to acquire an Update lock with Abort Blockers on the current key value, and an Insert Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RIn_U_LOW_PRIORITY	Task is waiting to acquire an Update lock with Low Priority on the current key value, and an Insert Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RIn_X	Occurs when a task is waiting to acquire an Exclusive lock on the current key value, and an Insert Range lock between the current and previous key.
LCK_M_RIn_X_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Exclusive lock with Abort Blockers on the current key value, and an Insert Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RIn_X_LOW_PRIORITY	Occurs when a task is waiting to acquire an Exclusive lock with Low Priority on the current key value, and an Insert Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RS_S	Occurs when a task is waiting to acquire a Shared lock on the current key value, and a Shared Range lock between the current and previous key.

TYPE	DESCRIPTION
LCK_M_RS_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared lock with Abort Blockers on the current key value, and a Shared Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RS_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared lock with Low Priority on the current key value, and a Shared Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RS_U	Occurs when a task is waiting to acquire an Update lock on the current key value, and an Update Range lock between the current and previous key.
LCK_M_RS_U_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Update lock with Abort Blockers on the current key value, and an Update Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RS_U_LOW_PRIORITY	Occurs when a task is waiting to acquire an Update lock with Low Priority on the current key value, and an Update Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RX_S	Occurs when a task is waiting to acquire a Shared lock on the current key value, and an Exclusive Range lock between the current and previous key.
LCK_M_RX_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared lock with Abort Blockers on the current key value, and an Exclusive Range with Abort Blockers lock between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RX_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared lock with Low Priority on the current key value, and an Exclusive Range with Low Priority lock between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.

TYPE	DESCRIPTION
LCK_M_RX_U	Occurs when a task is waiting to acquire an Update lock on the current key value, and an Exclusive range lock between the current and previous key.
LCK_M_RX_U_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Update lock with Abort Blockers on the current key value, and an Exclusive range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RX_U_LOW_PRIORITY	Occurs when a task is waiting to acquire an Update lock with Low Priority on the current key value, and an Exclusive range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RX_X	Occurs when a task is waiting to acquire an Exclusive lock on the current key value, and an Exclusive Range lock between the current and previous key.
LCK_M_RX_X_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Exclusive lock with Abort Blockers on the current key value, and an Exclusive Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_RX_X_LOW_PRIORITY	Occurs when a task is waiting to acquire an Exclusive lock with Low Priority on the current key value, and an Exclusive Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_S	Occurs when a task is waiting to acquire a Shared lock.
LCK_M_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_SCH_M	Occurs when a task is waiting to acquire a Schema Modify lock.

TYPE	DESCRIPTION
LCK_M_SCH_M_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Schema Modify lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_SCH_M_LOW_PRIORITY	Occurs when a task is waiting to acquire a Schema Modify lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_SCH_S	Occurs when a task is waiting to acquire a Schema Share lock.
LCK_M_SCH_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Schema Share lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_SCH_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a Schema Share lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_SIU	Occurs when a task is waiting to acquire a Shared With Intent Update lock.
LCK_M_SIU_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared With Intent Update lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_SIU_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared With Intent Update lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_SIX	Occurs when a task is waiting to acquire a Shared With Intent Exclusive lock.
LCK_M_SIX_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared With Intent Exclusive lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_SIX_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared With Intent Exclusive lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.

TYPE	DESCRIPTION
LCK_M_U	Occurs when a task is waiting to acquire an Update lock.
LCK_M_U_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Update lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_U_LOW_PRIORITY	Occurs when a task is waiting to acquire an Update lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_UIX	Occurs when a task is waiting to acquire an Update With Intent Exclusive lock.
LCK_M_UIX_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Update With Intent Exclusive lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_UIX_LOW_PRIORITY	Occurs when a task is waiting to acquire an Update With Intent Exclusive lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_X	Occurs when a task is waiting to acquire an Exclusive lock.
LCK_M_X_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Exclusive lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LCK_M_X_LOW_PRIORITY	Occurs when a task is waiting to acquire an Exclusive lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.), **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
LOG_POOL_SCAN	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
LOG_RATE_GOVERNOR	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
LOGBUFFER	Occurs when a task is waiting for space in the log buffer to store a log record. Consistently high values may indicate that the log devices cannot keep up with the amount of log being generated by the server.

TYPE	DESCRIPTION
LOGCAPTURE_LOGPOOLTRUNCPOINT	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
LOGGENERATION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
LOGMGR	Occurs when a task is waiting for any outstanding log I/Os to finish before shutting down the log while closing the database.
LOGMGR_FLUSH	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
LOGMGR_PMM_LOG	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
LOGMGR_QUEUE	Occurs while the log writer task waits for work requests.
LOGMGR_RESERVE_APPEND	Occurs when a task is waiting to see whether log truncation frees up log space to enable the task to write a new log record. Consider increasing the size of the log file(s) for the affected database to reduce this wait.
LOGPOOL_CACHESIZE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
LOGPOOL_CONSUMER	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
LOGPOOL_CONSUMERSET	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
LOGPOOL_FREEPOOLS	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
LOGPOOL_MGRSET	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
LOGPOOL_REPLACEMENTSET	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
LOGPOOLREFCOUNTEDOBJECT_REFDONE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
LOWFAIL_MEMMGR_QUEUE	Occurs while waiting for memory to be available for use.

TYPE	DESCRIPTION
MD_AGENT_YIELD	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
MD_LAZYCACHE_RWLOCK	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
MEMORY_ALLOCATION_EXT	Occurs while allocating memory from either the internal SQL Server memory pool or the operation system., **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
MEMORY_GRANT_UPDATE	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
METADATA_LAZYCACHE_RWLOCK	TBD **Applies to**: SQL Server 2008 R2 only.
MIGRATIONBUFFER	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
MISCELLANEOUS	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
MISCELLANEOUS	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
MSQL_DQ	Occurs when a task is waiting for a distributed query operation to finish. This is used to detect potential Multiple Active Result Set (MARS) application deadlocks. The wait ends when the distributed query call finishes.
MSQL_XACT_MGR_MUTEX	Occurs when a task is waiting to obtain ownership of the session transaction manager to perform a session level transaction operation.
MSQL_XACT_MUTEX	Occurs during synchronization of transaction usage. A request must acquire the mutex before it can use the transaction.
MSQL_XP	Occurs when a task is waiting for an extended stored procedure to end. SQL Server uses this wait state to detect potential MARS application deadlocks. The wait stops when the extended stored procedure call ends.
MSSEARCH	Occurs during Full-Text Search calls. This wait ends when the full-text operation completes. It does not indicate contention, but rather the duration of full-text operations.
NET_WAITFOR_PACKET	Occurs when a connection is waiting for a network packet during a network read.

TYPE	DESCRIPTION
NETWORKSXMLMGRLOAD	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
NODE_CACHE_MUTEX	TBD
OLEDB	Occurs when SQL Server calls the SQL Server Native Client OLE DB Provider. This wait type is not used for synchronization. Instead, it indicates the duration of calls to the OLE DB provider.
ONDEMAND_TASK_QUEUE	Occurs while a background task waits for high priority system task requests. Long wait times indicate that there have been no high priority requests to process, and should not cause concern.
PAGEIOLATCH_DT	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Destroy mode. Long waits may indicate problems with the disk subsystem.
PAGEIOLATCH_EX	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Exclusive mode. Long waits may indicate problems with the disk subsystem.
PAGEIOLATCH_KP	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Keep mode. Long waits may indicate problems with the disk subsystem.
PAGEIOLATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PAGEIOLATCH_SH	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Shared mode. Long waits may indicate problems with the disk subsystem.
PAGEIOLATCH_UP	Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Update mode. Long waits may indicate problems with the disk subsystem.
PAGELATCH_DT	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Destroy mode.
PAGELATCH_EX	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Exclusive mode.
PAGELATCH_KP	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Keep mode.
PAGELATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PAGELATCH_SH	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Shared mode.

TYPE	DESCRIPTION
PAGELATCH_UP	Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Update mode.
PARALLEL_BACKUP_QUEUE	Occurs when serializing output produced by RESTORE HEADERONLY, RESTORE FILELISTONLY, or RESTORE LABELONLY.
PARALLEL_REDO_DRAIN_WORKER	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PARALLEL_REDO_FLOW_CONTROL	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PARALLEL_REDO_LOG_CACHE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PARALLEL_REDO_TRAN_LIST	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PARALLEL_REDO_TRAN_TURN	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PARALLEL_REDO_WORKER_SYNC	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PARALLEL_REDO_WORKER_WAIT_WORK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PERFORMANCE_COUNTERS_RWLOCK	TBD
PHYSICAL_SEEDING_DMV	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
POOL_LOG_RATE_GOVERNOR	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PREEMPTIVE_ABR	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PREEMPTIVE_AUDIT_ACCESS_EVENTLOG	Occurs when the SQL Server Operating System (SQLOS) scheduler switches to preemptive mode to write an audit event to the Windows event log. **Applies to**: SQL Server 2008 R2 only.

TYPE	DESCRIPTION
PREEMPTIVE_AUDIT_ACCESS_SECLOG	Occurs when the SQLOS scheduler switches to preemptive mode to write an audit event to the Windows Security log. **Applies to**: SQL Server 2008 R2 only.
PREEMPTIVE_CLOSEBACKUPMEDIA	Occurs when the SQLOS scheduler switches to preemptive mode to close backup media.
PREEMPTIVE_CLOSEBACKUPTAPE	Occurs when the SQLOS scheduler switches to preemptive mode to close a tape backup device.
PREEMPTIVE_CLOSEBACKUPVDIDEVICE	Occurs when the SQLOS scheduler switches to preemptive mode to close a virtual backup device.
PREEMPTIVE_CLUSAPI_CLUSTERRESOURCECONTROL	Occurs when the SQLOS scheduler switches to preemptive mode to perform Windows failover cluster operations.
PREEMPTIVE_COM_COCREATEINSTANCE	Occurs when the SQLOS scheduler switches to preemptive mode to create a COM object.
PREEMPTIVE_COM_COGETCLASSOBJECT	TBD
PREEMPTIVE_COM_CREATEACCESSOR	TBD
PREEMPTIVE_COM_DELETEROWS	TBD
PREEMPTIVE_COM_GETCOMMANDTEXT	TBD
PREEMPTIVE_COM_GETDATA	TBD
PREEMPTIVE_COM_GETNEXTROWS	TBD
PREEMPTIVE_COM_GETRESULT	TBD
PREEMPTIVE_COM_GETROWSBYBOOKMARK	TBD
PREEMPTIVE_COM_LBFLUSH	TBD
PREEMPTIVE_COM_LBLOCKREGION	TBD
PREEMPTIVE_COM_LBREADAT	TBD
PREEMPTIVE_COM_LBSETSIZE	TBD
PREEMPTIVE_COM_LBSTAT	TBD
PREEMPTIVE_COM_LBUNLOCKREGION	TBD
PREEMPTIVE_COM_LBWRITEAT	TBD
PREEMPTIVE_COM_QUERYINTERFACE	TBD

TYPE	DESCRIPTION
PREEMPTIVE_COM_RELEASE	TBD
PREEMPTIVE_COM_RELEASEACCESSOR	TBD
PREEMPTIVE_COM_RELEASEROWS	TBD
PREEMPTIVE_COM_RELEASESESSION	TBD
PREEMPTIVE_COM_RESTARTPOSITION	TBD
PREEMPTIVE_COM_SEQSTRMREAD	TBD
PREEMPTIVE_COM_SEQSTRMREADANDWRITE	TBD
PREEMPTIVE_COM_SETDATAFAILURE	TBD
PREEMPTIVE_COM_SETPARAMETERINFO	TBD
PREEMPTIVE_COM_SETPARAMETERPROPERTIES	TBD
PREEMPTIVE_COM_STRMLOCKREGION	TBD
PREEMPTIVE_COM_STRMSEEKANDREAD	TBD
PREEMPTIVE_COM_STRMSEEKANDWRITE	TBD
PREEMPTIVE_COM_STRMSETSIZE	TBD
PREEMPTIVE_COM_STRMSTAT	TBD
PREEMPTIVE_COM_STRMUNLOCKREGION	TBD
PREEMPTIVE_CONSOLEWRITE	TBD
PREEMPTIVE_CREATEPARAM	TBD
PREEMPTIVE_DEBUG	TBD
PREEMPTIVE_DFSADDLINK	TBD
PREEMPTIVE_DFSLINKEXISTCHECK	TBD
PREEMPTIVE_DFSLINKHEALTHCHECK	TBD
PREEMPTIVE_DFSREMOVELINK	TBD
PREEMPTIVE_DFSREMOVEROOT	TBD
PREEMPTIVE_DFSROOTFOLDERCHECK	TBD

TYPE	DESCRIPTION
PREEMPTIVE_DFSROOTINIT	TBD
PREEMPTIVE_DFSROOTSHARECHECK	TBD
PREEMPTIVE_DTC_ABORT	TBD
PREEMPTIVE_DTC_ABORTREQUESTDONE	TBD
PREEMPTIVE_DTC_BEGINTRANSACTION	TBD
PREEMPTIVE_DTC_COMMITREQUESTDONE	TBD
PREEMPTIVE_DTC_ENLIST	TBD
PREEMPTIVE_DTC_PREPAREREQUESTDONE	TBD
PREEMPTIVE_FILESIZEGET	TBD
PREEMPTIVE_FSAOLEDB_ABORTTRANSACTION	TBD
PREEMPTIVE_FSAOLEDB_COMMITTRANSACTION	TBD
PREEMPTIVE_FSAOLEDB_STARTTRANSACTION	TBD
PREEMPTIVE_FSRECOVER_UNCONDITIONALUNDO	TBD
PREEMPTIVE_GETRMINFO	TBD
PREEMPTIVE_HADR_LEASE_MECHANISM	Always On Availability Groups lease manager scheduling for CSS diagnostics., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PREEMPTIVE_HTTP_EVENT_WAIT	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PREEMPTIVE_HTTP_REQUEST	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PREEMPTIVE_LOCKMONITOR	TBD
PREEMPTIVE_MSS_RELEASE	TBD
PREEMPTIVE_ODBCOPS	TBD
PREEMPTIVE_OLE_UNINIT	TBD
PREEMPTIVE_OLEDB_ABORTORCOMMITTRAN	TBD

TYPE	DESCRIPTION
PREEMPTIVE_OLEDB_ABORTTRAN	TBD
PREEMPTIVE_OLEDB_GETDATASOURCE	TBD
PREEMPTIVE_OLEDB_GETLITERALINFO	TBD
PREEMPTIVE_OLEDB_GETPROPERTIES	TBD
PREEMPTIVE_OLEDB_GETPROPERTYINFO	TBD
PREEMPTIVE_OLEDB_GETSCHEMALOCK	TBD
PREEMPTIVE_OLEDB_JOINTRANSACTION	TBD
PREEMPTIVE_OLEDB_RELEASE	TBD
PREEMPTIVE_OLEDB_SETPROPERTIES	TBD
PREEMPTIVE_OLEDBOPS	TBD
PREEMPTIVE_OS_ACCEPTSECURITYCONTEXT	TBD
PREEMPTIVE_OS_ACQUIRECREDENTIALSHANDLE	TBD
PREEMPTIVE_OS_AUTHENTICATIONOPS	TBD
PREEMPTIVE_OS_AUTHORIZATIONOPS	TBD
PREEMPTIVE_OS_AUTHZGETINFORMATIONFROMCONTEXT	TBD
PREEMPTIVE_OS_AUTHZINITIALIZECONTEXTFROMSID	TBD
PREEMPTIVE_OS_AUTHZINITIALIZERESOURCEMANAGER	TBD
PREEMPTIVE_OS_BACKUPREAD	TBD
PREEMPTIVE_OS_CLOSEHANDLE	TBD
PREEMPTIVE_OS_CLUSTEROPS	TBD
PREEMPTIVE_OS_COMOPS	TBD
PREEMPTIVE_OS_COMPLETEAUTHTOKEN	TBD
PREEMPTIVE_OS_COPYFILE	TBD
PREEMPTIVE_OS_CREATEDIRECTORY	TBD
PREEMPTIVE_OS_CREATEFILE	TBD

TYPE	DESCRIPTION
PREEMPTIVE_OS_CRYPTACQUIRECONTEXT	TBD
PREEMPTIVE_OS_CRYPTIMPORTKEY	TBD
PREEMPTIVE_OS_CRYPTOPS	TBD
PREEMPTIVE_OS_DECRYPTMESSAGE	TBD
PREEMPTIVE_OS_DELETEFILE	TBD
PREEMPTIVE_OS_DELETESECURITYCONTEXT	TBD
PREEMPTIVE_OS_DEVICEIOCONTROL	TBD
PREEMPTIVE_OS_DEVICEOPS	TBD
PREEMPTIVE_OS_DIRSVC_NETWORKOPS	TBD
PREEMPTIVE_OS_DISCONNECTNAMEDPIPE	TBD
PREEMPTIVE_OS_DOMAINSERVICESOPS	TBD
PREEMPTIVE_OS_DSGETDCNAME	TBD
PREEMPTIVE_OS_DTCOPS	TBD
PREEMPTIVE_OS_ENCRYPTMESSAGE	TBD
PREEMPTIVE_OS_FILEOPS	TBD
PREEMPTIVE_OS_FINDFILE	TBD
PREEMPTIVE_OS_FLUSHFILEBUFFERS	TBD
PREEMPTIVE_OS_FORMATMESSAGE	TBD
PREEMPTIVE_OS_FREECREDENTIALSHANDLE	TBD
PREEMPTIVE_OS_FREELIBRARY	TBD
PREEMPTIVE_OS_GENERICOPS	TBD
PREEMPTIVE_OS_GETADDRINFO	TBD
PREEMPTIVE_OS_GETCOMPRESSEDFILESIZE	TBD
PREEMPTIVE_OS_GETDISKFREESPACE	TBD
PREEMPTIVE_OS_GETFILEATTRIBUTES	TBD

TYPE	DESCRIPTION
PREEMPTIVE_OS_GETFILESIZE	TBD
PREEMPTIVE_OS_GETFINALFILEPATHBYHANDLE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PREEMPTIVE_OS_GETLONGPATHNAME	TBD
PREEMPTIVE_OS_GETPROCADDRESS	TBD
PREEMPTIVE_OS_GETVOLUMENAMEFORVOLUMEMOUNTPOINT	TBD
PREEMPTIVE_OS_GETVOLUMEPATHNAME	TBD
PREEMPTIVE_OS_INITIALIZESECURITYCONTEXT	TBD
PREEMPTIVE_OS_LIBRARYOPS	TBD
PREEMPTIVE_OS_LOADLIBRARY	TBD
PREEMPTIVE_OS_LOGONUSER	TBD
PREEMPTIVE_OS_LOOKUPACCOUNTSID	TBD
PREEMPTIVE_OS_MESSAGEQUEUEOPS	TBD
PREEMPTIVE_OS_MOVEFILE	TBD
PREEMPTIVE_OS_NETGROUPGETUSERS	TBD
PREEMPTIVE_OS_NETLOCALGROUPGETMEMBERS	TBD
PREEMPTIVE_OS_NETUSERGETGROUPS	TBD
PREEMPTIVE_OS_NETUSERGETLOCALGROUPS	TBD
PREEMPTIVE_OS_NETUSERMODALSGET	TBD
PREEMPTIVE_OS_NETVALIDATEPASSWORDPOLICY	TBD
PREEMPTIVE_OS_NETVALIDATEPASSWORDPOLICYFREE	TBD
PREEMPTIVE_OS_OPENDIRECTORY	TBD
PREEMPTIVE_OS_PDH_WMI_INIT	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PREEMPTIVE_OS_PIPEOPS	TBD

TYPE	DESCRIPTION
PREEMPTIVE_OS_PROCESSOPS	TBD
PREEMPTIVE_OS_QUERYCONTEXTATTRIBUTES	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PREEMPTIVE_OS_QUERYREGISTRY	TBD
PREEMPTIVE_OS_QUERYSECURITYCONTEXTTOKEN	TBD
PREEMPTIVE_OS_REMOVEDIRECTORY	TBD
PREEMPTIVE_OS_REPORTEVENT	TBD
PREEMPTIVE_OS_REVERTTOSELF	TBD
PREEMPTIVE_OS_RSFXDEVICEOPS	TBD
PREEMPTIVE_OS_SECURITYOPS	TBD
PREEMPTIVE_OS_SERVICEOPS	TBD
PREEMPTIVE_OS_SETENDOFFILE	TBD
PREEMPTIVE_OS_SETFILEPOINTER	TBD
PREEMPTIVE_OS_SETFILEVALIDDATA	TBD
PREEMPTIVE_OS_SETNAMEDSECURITYINFO	TBD
PREEMPTIVE_OS_SQLCLROPS	TBD
PREEMPTIVE_OS_SQMLAUNCH	TBD **Applies to**: SQL Server 2008 R2 through SQL Server 2016 (13.x).
PREEMPTIVE_OS_VERIFYSIGNATURE	TBD
PREEMPTIVE_OS_VERIFYTRUST	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PREEMPTIVE_OS_VSSOPS	TBD
PREEMPTIVE_OS_WAITFORSINGLEOBJECT	TBD
PREEMPTIVE_OS_WINSOCKOPS	TBD
PREEMPTIVE_OS_WRITEFILE	TBD
PREEMPTIVE_OS_WRITEFILEGATHER	TBD

TYPE	DESCRIPTION
PREEMPTIVE_OS_WSASETLASTERROR	TBD
PREEMPTIVE_REENLIST	TBD
PREEMPTIVE_RESIZELOG	TBD
PREEMPTIVE_ROLLFORWARDREDO	TBD
PREEMPTIVE_ROLLFORWARDUNDO	TBD
PREEMPTIVE_SB_STOPENDPOINT	TBD
PREEMPTIVE_SERVER_STARTUP	TBD
PREEMPTIVE_SETRMINFO	TBD
PREEMPTIVE_SHAREDMEM_GETDATA	TBD
PREEMPTIVE_SNIOPEN	TBD
PREEMPTIVE_SOSHOST	TBD
PREEMPTIVE_SOSTESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PREEMPTIVE_SP_SERVER_DIAGNOSTICS	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PREEMPTIVE_STARTRM	TBD
PREEMPTIVE_STREAMFCB_CHECKPOINT	TBD
PREEMPTIVE_STREAMFCB_RECOVER	TBD
PREEMPTIVE_STRESSDRIVER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PREEMPTIVE_TESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PREEMPTIVE_TRANSIMPORT	TBD
PREEMPTIVE_UNMARSHALPROPAGATIONTOKEN	TBD
PREEMPTIVE_VSS_CREATESNAPSHOT	TBD
PREEMPTIVE_VSS_CREATEVOLUMESNAPSHOT	TBD
PREEMPTIVE_XE_CALLBACKEXECUTE	TBD

TYPE	DESCRIPTION
PREEMPTIVE_XE_CX_FILE_OPEN	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
PREEMPTIVE_XE_CX_HTTP_CALL	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
PREEMPTIVE_XE_DISPATCHER	TBD
PREEMPTIVE_XE_ENGINEINIT	TBD
PREEMPTIVE_XE_GETTARGETSTATE	TBD
PREEMPTIVE_XE_SESSIONCOMMIT	TBD
PREEMPTIVE_XE_TARGETFINALIZE	TBD
PREEMPTIVE_XE_TARGETINIT	TBD
PREEMPTIVE_XE_TIMERRUN	TBD
PREEMPTIVE_XETESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
PRINT_ROLLBACK_PROGRESS	Used to wait while user processes are ended in a database that has been transitioned by using the ALTER DATABASE termination clause. For more information, see ALTER DATABASE (Transact-SQL).
PRU_ROLLBACK_DEFERRED	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_ALL_COMPONENTS_INITIALIZED	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_COOP_SCAN	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_DIRECTLOGCONSUMER_GETNEXT	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PWAIT_EVENT_SESSION_INIT_MUTEX	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_FABRIC_REPLICA_CONTROLLER_DATA_LOSS	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

TYPE	DESCRIPTION
PWAIT_HADR_ACTION_COMPLETED	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_HADR_CHANGE_NOTIFIER_TERMINATION_SYNC	Occurs when a background task is waiting for the termination of the background task that receives (via polling) Windows Server Failover Clustering notifications., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_HADR_CLUSTER_INTEGRATION	An append, replace, and/or remove operation is waiting to grab a write lock on an Always On internal list (such as a list of networks, network addresses, or availability group listeners). Internal use only, **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_HADR_FAILOVER_COMPLETED	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_HADR_JOIN	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
PWAIT_HADR_OFFLINE_COMPLETED	An Always On drop availability group operation is waiting for the target availability group to go offline before destroying Windows Server Failover Clustering objects., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_HADR_ONLINE_COMPLETED	An Always On create or failover availability group operation is waiting for the target availability group to come online., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_HADR_POST_ONLINE_COMPLETED	An Always On drop availability group operation is waiting for the termination of any background task that was scheduled as part of a previous command. For example, there may be a background task that is transitioning availability databases to the primary role. The DROP AVAILABILITY GROUP DDL must wait for this background task to terminate in order to avoid race conditions., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_HADR_SERVER_READY_CONNECTIONS	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_HADR_WORKITEM_COMPLETED	Internal wait by a thread waiting for an async work task to complete. This is an expected wait and is for CSS use., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
PWAIT_HADRSIM	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
PWAIT_LOG_CONSOLIDATION_IO	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
PWAIT_LOG_CONSOLIDATION_POLL	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
PWAIT_MD_LOGIN_STATS	Occurs during internal synchronization in metadata on login stats., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_MD_RELATION_CACHE	Occurs during internal synchronization in metadata on table or index., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_MD_SERVER_CACHE	Occurs during internal synchronization in metadata on linked servers., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_MD_UPGRADE_CONFIG	Occurs during internal synchronization in upgrading server wide configurations., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_PREEMPTIVE_APP_USAGE_TIMER	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
PWAIT_PREEMPTIVE_AUDIT_ACCESS_WINDOWSLOG	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_QRY_BPMEMORY	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_REPLICA_ONLINE_INIT_MUTEX	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_RESOURCE_SEMAPHORE_FT_PARALLEL_QUERY_SYNC	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
PWAIT_SBS_FILE_OPERATION	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.

TYPE	DESCRIPTION
PWAIT_XTP_FSSTORAGE_MAINTENANCE	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
PWAIT_XTP_HOST_STORAGE_WAIT	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_ASYNC_CHECK_CONSISTENCY_TASK	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_ASYNC_PERSIST_TASK	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_ASYNC_PERSIST_TASK_START	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_ASYNC_QUEUE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
QDS_BCKG_TASK	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_BLOOM_FILTER	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
QDS_CLEANUP_STALE_QUERIES_TASK_MAIN_LOOP_SLEEP	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_CTXS	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_DB_DISK	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_DYN_VECTOR	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_EXCLUSIVE_ACCESS	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
QDS_HOST_INIT	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.

TYPE	DESCRIPTION
QDS_LOADDB	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_PERSIST_TASK_MAIN_LOOP_SLEEP	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_QDS_CAPTURE_INIT	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
QDS_SHUTDOWN_QUEUE	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_STMT	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_STMT_DISK	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_TASK_SHUTDOWN	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QDS_TASK_START	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QE_WARN_LIST_SYNC	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
QPJOB_KILL	Indicates that an asynchronous automatic statistics update was canceled by a call to KILL as the update was starting to run. The terminating thread is suspended, waiting for it to start listening for KILL commands. A good value is less than one second.
QPJOB_WAITFOR_ABORT	Indicates that an asynchronous automatic statistics update was canceled by a call to KILL when it was running. The update has now completed but is suspended until the terminating thread message coordination is complete. This is an ordinary but rare state, and should be very short. A good value is less than one second.
QRY_MEM_GRANT_INFO_MUTEX	Occurs when Query Execution memory management tries to control access to static grant information list. This state lists information about the current granted and waiting memory requests. This state is a simple access control state. There should never be a long wait on this state. If this mutex is not released, all new memory-using queries will stop responding.

TYPE	DESCRIPTION
QRY_PARALLEL_THREAD_MUTEX	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
QRY_PROFILE_LIST_MUTEX	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
QUERY_ERRHDL_SERVICE_DONE	Identified for informational purposes only. Not supported. **Applies to**: SQL Server 2008 R2 only.
QUERY_WAIT_ERRHDL_SERVICE	Identified for informational purposes only. Not supported. **Applies to**: SQL Server 2008 R2 only.
QUERY_EXECUTION_INDEX_SORT_EVENT_OPEN	Occurs in certain cases when offline create index build is run in parallel, and the different worker threads that are sorting synchronize access to the sort files.
QUERY_NOTIFICATION_MGR_MUTEX	Occurs during synchronization of the garbage collection queue in the Query Notification Manager.
QUERY_NOTIFICATION_SUBSCRIPTION_MUTEX	Occurs during state synchronization for transactions in Query Notifications.
QUERY_NOTIFICATION_TABLE_MGR_MUTEX	Occurs during internal synchronization within the Query Notification Manager.
QUERY_NOTIFICATION_UNITTEST_MUTEX	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
QUERY_OPTIMIZER_PRINT_MUTEX	Occurs during synchronization of query optimizer diagnostic output production. This wait type only occurs if diagnostic settings have been enabled under direction of Microsoft Product Support.
QUERY_TASK_ENQUEUE_MUTEX	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
QUERY_TRACEOUT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
RBIO_WAIT_VLF	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
RECOVER_CHANGEDB	Occurs during synchronization of database status in warm standby database.
RECOVERY_MGR_LOCK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

TYPE	DESCRIPTION
REDO_THREAD_PENDING_WORK	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
REDO_THREAD_SYNC	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
REMOTE_BLOCK_IO	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
REMOTE_DATA_ARCHIVE_MIGRATION_DMV	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
REMOTE_DATA_ARCHIVE_SCHEMA_DMV	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
REMOTE_DATA_ARCHIVE_SCHEMA_TASK_QUEUE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
REPL_CACHE_ACCESS	Occurs during synchronization on a replication article cache. During these waits, the replication log reader stalls, and data definition language (DDL) statements on a published table are blocked.
REPL_HISTORYCACHE_ACCESS	TBD
REPL_SCHEMA_ACCESS	Occurs during synchronization of replication schema version information. This state exists when DDL statements are executed on the replicated object, and when the log reader builds or consumes versioned schema based on DDL occurrence. Contention can be seen on this wait type if you have many published databases on a single publisher with transactional replication and the published databases are very active.
REPL_TRANFSINFO_ACCESS	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
REPL_TRANHASHTABLE_ACCESS	TBD
REPL_TRANTEXTINFO_ACCESS	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
REPLICA_WRITES	Occurs while a task waits for completion of page writes to database snapshots or DBCC replicas.

TYPE	DESCRIPTION
REQUEST_DISPENSER_PAUSE	Occurs when a task is waiting for all outstanding I/O to complete, so that I/O to a file can be frozen for snapshot backup.
REQUEST_FOR_DEADLOCK_SEARCH	Occurs while the deadlock monitor waits to start the next deadlock search. This wait is expected between deadlock detections, and lengthy total waiting time on this resource does not indicate a problem.
RESERVED_MEMORY_ALLOCATION_EXT	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
RESMGR_THROTTLED	Occurs when a new request comes in and is throttled based on the GROUP_MAX_REQUESTS setting.
RESOURCE_GOVERNOR_IDLE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
RESOURCE_QUEUE	Occurs during synchronization of various internal resource queues.
RESOURCE_SEMAPHORE	Occurs when a query memory request cannot be granted immediately due to other concurrent queries. High waits and wait times may indicate excessive number of concurrent queries, or excessive memory request amounts.
RESOURCE_SEMAPHORE_MUTEX	Occurs while a query waits for its request for a thread reservation to be fulfilled. It also occurs when synchronizing query compile and memory grant requests.
RESOURCE_SEMAPHORE_QUERY_COMPILE	Occurs when the number of concurrent query compilations reaches a throttling limit. High waits and wait times may indicate excessive compilations, recompiles, or uncachable plans.
RESOURCE_SEMAPHORE_SMALL_QUERY	Occurs when memory request by a small query cannot be granted immediately due to other concurrent queries. Wait time should not exceed more than a few seconds, because the server transfers the request to the main query memory pool if it fails to grant the requested memory within a few seconds. High waits may indicate an excessive number of concurrent small queries while the main memory pool is blocked by waiting queries. **Applies to**: SQL Server 2008 R2 only.
RESTORE_FILEHANDLECACHE_ENTRYLOCK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
RESTORE_FILEHANDLECACHE_LOCK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
RG_RECONFIG	TBD

TYPE	DESCRIPTION
ROWGROUP_OP_STATS	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
ROWGROUP_VERSION	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
RTDATA_LIST	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
SATELLITE_CARGO	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SATELLITE_SERVICE_SETUP	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SATELLITE_TASK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SBS_DISPATCH	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
SBS_RECEIVE_TRANSPORT	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
SBS_TRANSPORT	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
SCAN_CHAR_HASH_ARRAY_INITIALIZATION	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SEC_DROP_TEMP_KEY	Occurs after a failed attempt to drop a temporary security key before a retry attempt.
SECURITY_CNG_PROVIDER_MUTEX	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
SECURITY_CRYPTO_CONTEXT_MUTEX	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SECURITY_DBE_STATE_MUTEX	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.

TYPE	DESCRIPTION
SECURITY_KEYRING_RWLOCK	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SECURITY_MUTEX	Occurs when there is a wait for mutexes that control access to the global list of Extensible Key Management (EKM) cryptographic providers and the session-scoped list of EKM sessions.
SECURITY_RULETABLE_MUTEX	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SEMPLAT_DSI_BUILD	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SEQUENCE_GENERATION	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SEQUENTIAL_GUID	Occurs while a new sequential GUID is being obtained.
SERVER_IDLE_CHECK	Occurs during synchronization of SQL Server instance idle status when a resource monitor is attempting to declare a SQL Server instance as idle or trying to wake up.
SERVER_RECONFIGURE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SESSION_WAIT_STATS_CHILDREN	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SHARED_DELTASTORE_CREATION	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SHUTDOWN	Occurs while a shutdown statement waits for active connections to exit.
SLEEP_BPOOL_FLUSH	Occurs when a checkpoint is throttling the issuance of new I/Os in order to avoid flooding the disk subsystem.
SLEEP_BUFFERPOOL_HELPLW	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SLEEP_DBSTARTUP	Occurs during database startup while waiting for all databases to recover.
SLEEP_DCOMSTARTUP	Occurs once at most during SQL Server instance startup while waiting for DCOM initialization to complete.

TYPE	DESCRIPTION
SLEEP_MASTERDBREADY	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SLEEP_MASTERMDREADY	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SLEEP_MASTERUPGRADED	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SLEEP_MEMORYPOOL_ALLOCATEPAGES	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SLEEP_MSDBSTARTUP	Occurs when SQL Trace waits for the msdb database to complete startup.
SLEEP_RETRY_VIRTUALALLOC	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SLEEP_SYSTEMTASK	Occurs during the start of a background task while waiting for tempdb to complete startup.
SLEEP_TASK	Occurs when a task sleeps while waiting for a generic event to occur.
SLEEP_TEMPDBSTARTUP	Occurs while a task waits for tempdb to complete startup.
SLEEP_WORKSPACE_ALLOCATEPAGE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SLO_UPDATE	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
SMSYNC	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SNI_CONN_DUP	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
SNI_CRITICAL_SECTION	Occurs during internal synchronization within SQL Server networking components.
SNI_HTTP_WAITFOR_0_DISCON	Occurs during SQL Server shutdown, while waiting for outstanding HTTP connections to exit.

TYPE	DESCRIPTION
SNI_LISTENER_ACCESS	Occurs while waiting for non-uniform memory access (NUMA) nodes to update state change. Access to state change is serialized.
SNI_TASK_COMPLETION	Occurs when there is a wait for all tasks to finish during a NUMA node state change.
SNI_WRITE_ASYNC	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
SOAP_READ	Occurs while waiting for an HTTP network read to complete.
SOAP_WRITE	Occurs while waiting for an HTTP network write to complete.
SOCKETDUPLICATEQUEUE_CLEANUP	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
SOS_CALLBACK_REMOVAL	Occurs while performing synchronization on a callback list in order to remove a callback. It is not expected for this counter to change after server initialization is completed.
SOS_DISPATCHER_MUTEX	Occurs during internal synchronization of the dispatcher pool. This includes when the pool is being adjusted.
SOS_LOCALALLOCATORLIST	Occurs during internal synchronization in the SQL Server memory manager. **Applies to**: SQL Server 2008 R2 only.
SOS_MEMORY_TOPLEVELBLOCKALLOCATOR	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SOS_MEMORY_USAGE_ADJUSTMENT	Occurs when memory usage is being adjusted among pools.
SOS_OBJECT_STORE_DESTROY_MUTEX	Occurs during internal synchronization in memory pools when destroying objects from the pool.
SOS_PHYS_PAGE_CACHE	Accounts for the time a thread waits to acquire the mutex it must acquire before it allocates physical pages or before it returns those pages to the operating system. Waits on this type only appear if the instance of SQL Server uses AWE memory., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SOS_PROCESS_AFFINITY_MUTEX	Occurs during synchronizing of access to process affinity settings.
SOS_RESERVEDMEMBLOCKLIST	Occurs during internal synchronization in the SQL Server memory manager. **Applies to**: SQL Server 2008 R2 only.

TYPE	DESCRIPTION
SOS_SCHEDULER_YIELD	Occurs when a task voluntarily yields the scheduler for other tasks to execute. During this wait the task is waiting for its quantum to be renewed.
SOS_SMALL_PAGE_ALLOC	Occurs during the allocation and freeing of memory that is managed by some memory objects.
SOS_STACKSTORE_INIT_MUTEX	Occurs during synchronization of internal store initialization.
SOS_SYNC_TASK_ENQUEUE_EVENT	Occurs when a task is started in a synchronous manner. Most tasks in SQL Server are started in an asynchronous manner, in which control returns to the starter immediately after the task request has been placed on the work queue.
SOS_VIRTUALMEMORY_LOW	Occurs when a memory allocation waits for a resource manager to free up virtual memory.
SOSHOST_EVENT	Occurs when a hosted component, such as CLR, waits on a SQL Server event synchronization object.
SOSHOST_INTERNAL	Occurs during synchronization of memory manager callbacks used by hosted components, such as CLR.
SOSHOST_MUTEX	Occurs when a hosted component, such as CLR, waits on a SQL Server mutex synchronization object.
SOSHOST_RWLOCK	Occurs when a hosted component, such as CLR, waits on a SQL Server reader-writer synchronization object.
SOSHOST_SEMAPHORE	Occurs when a hosted component, such as CLR, waits on a SQL Server semaphore synchronization object.
SOSHOST_SLEEP	Occurs when a hosted task sleeps while waiting for a generic event to occur. Hosted tasks are used by hosted components such as CLR.
SOSHOST_TRACELOCK	Occurs during synchronization of access to trace streams.
SOSHOST_WAITFORDONE	Occurs when a hosted component, such as CLR, waits for a task to complete.
SP_PREEMPTIVE_SERVER_DIAGNOSTICS_SLEEP	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SP_SERVER_DIAGNOSTICS_BUFFER_ACCESS	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SP_SERVER_DIAGNOSTICS_INIT_MUTEX	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
SP_SERVER_DIAGNOSTICS_SLEEP	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SQLCLR_APPDOMAIN	Occurs while CLR waits for an application domain to complete startup.
SQLCLR_ASSEMBLY	Occurs while waiting for access to the loaded assembly list in the appdomain.
SQLCLR_DEADLOCK_DETECTION	Occurs while CLR waits for deadlock detection to complete.
SQLCLR_QUANTUM_PUNISHMENT	Occurs when a CLR task is throttled because it has exceeded its execution quantum. This throttling is done in order to reduce the effect of this resource-intensive task on other tasks.
SQLSORT_NORMMUTEX	Occurs during internal synchronization, while initializing internal sorting structures.
SQLSORT_SORTMUTEX	Occurs during internal synchronization, while initializing internal sorting structures.
SQLTRACE_BUFFER_FLUSH	Occurs when a task is waiting for a background task to flush trace buffers to disk every four seconds. **Applies to**: SQL Server 2008 R2 only.
SQLTRACE_FILE_BUFFER	Occurs during synchronization on trace buffers during a file trace., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SQLTRACE_FILE_READ_IO_COMPLETION	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SQLTRACE_FILE_WRITE_IO_COMPLETION	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SQLTRACE_INCREMENTAL_FLUSH_SLEEP	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SQLTRACE_LOCK	TBD **APPLIES TO**: SQL Server 2008 R2 only.
SQLTRACE_PENDING_BUFFER_WRITERS	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
SQLTRACE_SHUTDOWN	Occurs while trace shutdown waits for outstanding trace events to complete.

TYPE	DESCRIPTION
SQLTRACE_WAIT_ENTRIES	Occurs while a SQL Trace event queue waits for packets to arrive on the queue.
SRVPROC_SHUTDOWN	Occurs while the shutdown process waits for internal resources to be released to shutdown cleanly.
STARTUP_DEPENDENCY_MANAGER	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
TDS_BANDWIDTH_STATE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
TDS_INIT	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
TDS_PROXY_CONTAINER	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
TEMPOBJ	Occurs when temporary object drops are synchronized. This wait is rare, and only occurs if a task has requested exclusive access for temp table drops.
TEMPORAL_BACKGROUND_PROCEED_CLEANUP	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
TERMINATE_LISTENER	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
THREADPOOL	Occurs when a task is waiting for a worker to run on. This can indicate that the maximum worker setting is too low, or that batch executions are taking unusually long, thus reducing the number of workers available to satisfy other batches.
TIMEPRIV_TIMEPERIOD	Occurs during internal synchronization of the Extended Events timer.
TRACE_EVTNOTIF	TBD
TRACEWRITE	Occurs when the SQL Trace rowset trace provider waits for either a free buffer or a buffer with events to process.
TRAN_MARKLATCH_DT	Occurs when waiting for a destroy mode latch on a transaction mark latch. Transaction mark latches are used for synchronization of commits with marked transactions.
TRAN_MARKLATCH_EX	Occurs when waiting for an exclusive mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions.

TYPE	DESCRIPTION
TRAN_MARKLATCH_KP	Occurs when waiting for a keep mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions.
TRAN_MARKLATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
TRAN_MARKLATCH_SH	Occurs when waiting for a shared mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions.
TRAN_MARKLATCH_UP	Occurs when waiting for an update mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions.
TRANSACTION_MUTEX	Occurs during synchronization of access to a transaction by multiple batches.
UCS_ENDPOINT_CHANGE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
UCS_MANAGER	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
UCS_MEMORY_NOTIFICATION	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
UCS_SESSION_REGISTRATION	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
UCS_TRANSPORT	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
UCS_TRANSPORT_STREAM_CHANGE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
UTIL_PAGE_ALLOC	Occurs when transaction log scans wait for memory to be available during memory pressure.
VDI_CLIENT_COMPLETECOMMAND	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
VDI_CLIENT_GETCOMMAND	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.

TYPE	DESCRIPTION
VDI_CLIENT_OPERATION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
VDI_CLIENT_OTHER	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
VERSIONING_COMMITTING	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
VIA_ACCEPT	Occurs when a Virtual Interface Adapter (VIA) provider connection is completed during startup.
VIEW_DEFINITION_MUTEX	Occurs during synchronization on access to cached view definitions.
WAIT_FOR_RESULTS	Occurs when waiting for a query notification to be triggered.
WAIT_SCRIPTDEPLOYMENT_REQUEST	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_SCRIPTDEPLOYMENT_WORKER	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XLOGREAD_SIGNAL	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
WAIT_XTP_ASYNC_TX_COMPLETION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_CKPT_AGENT_WAKEUP	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_CKPT_CLOSE	Occurs when waiting for a checkpoint to complete., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_CKPT_ENABLED	Occurs when checkpointing is disabled, and waiting for checkpointing to be enabled., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_CKPT_STATE_LOCK	Occurs when synchronizing checking of checkpoint state., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.

TYPE	DESCRIPTION
WAIT_XTP_COMPILE_WAIT	TBD **APPLIES TO**: SQL Server 2017 (14.x) through SQL Server 2017.
WAIT_XTP_GUEST	Occurs when the database memory allocator needs to stop receiving low-memory notifications., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
WAIT_XTP_HOST_WAIT	Occurs when waits are triggered by the database engine and implemented by the host., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_OFFLINE_CKPT_BEFORE_REDO	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_OFFLINE_CKPT_LOG_IO	Occurs when offline checkpoint is waiting for a log read IO to complete., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_OFFLINE_CKPT_NEW_LOG	Occurs when offline checkpoint is waiting for new log records to scan., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_PROCEDURE_ENTRY	Occurs when a drop procedure is waiting for all current executions of that procedure to complete., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_RECOVERY	Occurs when database recovery is waiting for recovery of memory-optimized objects to finish., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WAIT_XTP_SERIAL_RECOVERY	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
WAIT_XTP_SWITCH_TO_INACTIVE	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
WAIT_XTP_TASK_SHUTDOWN	Occurs when waiting for an In-Memory OLTP thread to complete., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
WAIT_XTP_TRAN_DEPENDENCY	Occurs when waiting for transaction dependencies., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.

TYPE	DESCRIPTION
WAITFOR	Occurs as a result of a WAITFOR Transact-SQL statement. The duration of the wait is determined by the parameters to the statement. This is a user-initiated wait.
WAITFOR_PER_QUEUE	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
WAITFOR_TASKSHUTDOWN	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
WAITSTAT_MUTEX	Occurs during synchronization of access to the collection of statistics used to populate sys.dm_os_wait_stats.
WCC	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
WINDOW_AGGREGATES_MULTIPASS	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
WINFAB_API_CALL	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WINFAB_REPLICA_BUILD_OPERATION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
WINFAB_REPORT_FAULT	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
WORKTBL_DROP	Occurs while pausing before retrying, after a failed worktable drop.
WRITE_COMPLETION	Occurs when a write operation is in progress.
WRITELOG	Occurs while waiting for a log flush to complete. Common operations that cause log flushes are checkpoints and transaction commits.
XACT_OWN_TRANSACTION	Occurs while waiting to acquire ownership of a transaction.
XACT_RECLAIM_SESSION	Occurs while waiting for the current owner of a session to release ownership of the session.
XACTLOCKINFO	Occurs during synchronization of access to the list of locks for a transaction. In addition to the transaction itself, the list of locks is accessed by operations such as deadlock detection and lock migration during page splits.
XACTWORKSPACE_MUTEX	Occurs during synchronization of defections from a transaction, as well as the number of database locks between enlist members of a transaction.

TYPE	DESCRIPTION
XDB_CONN_DUP_HASH	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
XDES_HISTORY	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
XDES_OUT_OF_ORDER_LIST	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
XDES_SNAPSHOT	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
XDESTSVERMGR	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
XE_BUFFERMGR_ALLPROCESSED_EVENT	Occurs when Extended Events session buffers are flushed to targets. This wait occurs on a background thread.
XE_BUFFERMGR_FREEBUF_EVENT	Occurs when either of the following conditions is true:
XE_CALLBACK_LIST	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.
XE_CX_FILE_READ	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
XE_DISPATCHER_CONFIG_SESSION_LIST	Occurs when an Extended Events session that is using asynchronous targets is started or stopped. This wait indicates either of the following:
XE_DISPATCHER_JOIN	Occurs when a background thread that is used for Extended Events sessions is terminating.
XE_DISPATCHER_WAIT	Occurs when a background thread that is used for Extended Events sessions is waiting for event buffers to process.
XE_FILE_TARGET_TVF	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
XE_LIVE_TARGET_TVF	TBD **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

TYPE	DESCRIPTION
XE_MODULEMGR_SYNC	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
XE_OLS_LOCK	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
XE_PACKAGE_LOCK_BACKOFF	Identified for informational purposes only. Not supported. **Applies to**: SQL Server 2008 R2 only.
XE_SERVICES_EVENTMANUAL	TBD
XE_SERVICES_MUTEX	TBD
XE_SERVICES_RWLOCK	TBD
XE_SESSION_CREATE_SYNC	TBD
XE_SESSION_FLUSH	TBD
XE_SESSION_SYNC	TBD
XE_STM_CREATE	TBD
XE_TIMER_EVENT	TBD
XE_TIMER_MUTEX	TBD
XE_TIMER_TASK_DONE	TBD
XIO_CREDENTIAL_MGR_RWLOCK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
XIO_CREDENTIAL_RWLOCK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
XIO_EDS_MGR_RWLOCK	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
XIO_EDS_RWLOCK	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
XIO_IOSTATS_BLOBLIST_RWLOCK	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.
XIO_IOSTATS_FCBLIST_RWLOCK	TBD **Applies to**: SQL Server 2017 (14.x) through SQL Server 2017.

TYPE	DESCRIPTION
XIO_LEASE_RENEW_MGR_RWLOCK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
XTP_HOST_DB_COLLECTION	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
XTP_HOST_LOG_ACTIVITY	TBD **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
XTP_HOST_PARALLEL_RECOVERY	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
XTP_PREEMPTIVE_TASK	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
XTP_TRUNCATION_LSN	TBD **Applies to**: SQL Server 2016 (13.x) through SQL Server 2017.
XTPPROC_CACHE_ACCESS	Occurs when for accessing all natively compiled stored procedure cache objects., **Applies to**: SQL Server 2014 (12.x) through SQL Server 2017.
XTPPROC_PARTITIONED_STACK_CREATE	Occurs when allocating per-NUMA node natively compiled stored procedure cache structures (must be done single threaded) for a given procedure., **Applies to**: SQL Server 2012 (11.x) through SQL Server 2017.

The following XEvents are related to partition **SWITCH** and online index rebuild. For information about syntax, see ALTER TABLE (Transact-SQL) and ALTER INDEX (Transact-SQL).

- lock_request_priority_state
- process_killed_by_abort_blockers
- ddl_with_wait_at_low_priority

  For a lock compatibility matrix, see sys.dm_tran_locks (Transact-SQL).

See also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)
sys.dm_exec_session_wait_stats (Transact-SQL)
sys.dm_db_wait_stats (Azure SQL Database)

sys.dm_os_windows_info (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns one row that displays Windows operating system version information.

Only applies to SQL Server running on Windows. To see similar informaton for SQL Server running on a non-Windows host, such as Linux, use sys.dm_os_host_info (Transact-SQL).

COLUMN NAME	DATA TYPE	DESCRIPTION
windows_release	**nvarchar(256)**	For Windows, returns the release number. For a list of values and descriptions, see Operating System Version (Windows). Cannot be NULL.
windows_service_pack_level	**nvarchar(256)**	For Windows, returns the service pack number. Cannot be NULL.
windows_sku	**int**	For Windows, returns the Windows Stock Keeping Unit (SKU) ID. For a list of SKU IDs and descriptions, see GetProductInfo Function. Is NULLable.
os_language_version	**int**	For Windows, returns the Windows locale identifier (LCID) of the operating system. For a list of LCID values and descriptions, see Locale IDs Assigned by Microsoft. Cannot be NULL.

Permissions

The SELECT permission on sys.dm_os_windows_info is granted to the public role by default. If revoked, requires VIEW SERVER STATE permission on the server.

Limitations and Restrictions

To see informaton for SQL running on a non-Windows host, such as Linux, use sys.dm_os_host_info (Transact-SQL).

Examples

The following example returns all columns from the **sys.dm_os_windows_info** view.

```
SELECT windows_release, windows_service_pack_level, windows_sku, os_language_version
FROM sys.dm_os_windows_info;
```

Here is a sample result set.

`windows_release windows_service_pack_level windows_sku os_language_version`

```
--------------- --------------------------- ------------ -------------------
```

```
6.0 Service Pack 2 4 1033
```

See Also

sys.dm_os_sys_info (Transact-SQL)
sys.dm_os_host_info

sys.dm_os_workers (Transact-SQL)

5/4/2018 • 5 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a row for every worker in the system.

> **NOTE**
>
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_os_workers**.

COLUMN NAME	DATA TYPE	DESCRIPTION
worker_address	**varbinary(8)**	Memory address of the worker.
status	**int**	Internal use only.
is_preemptive	**bit**	1 = Worker is running with preemptive scheduling. Any worker that is running external code is run under preemptive scheduling.
is_fiber	**bit**	1 = Worker is running with lightweight pooling. For more information, see sp_configure (Transact-SQL).
is_sick	**bit**	1 = Worker is stuck trying to obtain a spin lock. If this bit is set, this might indicate a problem with contention on a frequently accessed object.
is_in_cc_exception	**bit**	1 = Worker is currently handling a non- SQL Server exception.
is_fatal_exception	**bit**	Specifies whether this worker received a fatal exception.
is_inside_catch	**bit**	1 = Worker is currently handling an exception.
is_in_polling_io_completion_routine	**bit**	1 = Worker is currently running an I/O completion routine for a pending I/O. For more information, see sys.dm_io_pending_io_requests (Transact-SQL).
context_switch_count	**int**	Number of scheduler context switches that are performed by this worker.
pending_io_count	**int**	Number of physical I/Os that are performed by this worker.

COLUMN NAME	DATA TYPE	DESCRIPTION
pending_io_byte_count	**bigint**	Total number of bytes for all pending physical I/Os for this worker.
pending_io_byte_average	**int**	Average number of bytes for physical I/Os for this worker.
wait_started_ms_ticks	**bigint**	Point in time, in ms_ticks, when this worker entered the SUSPENDED state. Subtracting this value from ms_ticks in sys.dm_os_sys_info returns the number of milliseconds that the worker has been waiting.
wait_resumed_ms_ticks	**bigint**	Point in time, in ms_ticks, when this worker entered the RUNNABLE state. Subtracting this value from ms_ticks in sys.dm_os_sys_info returns the number of milliseconds that the worker has been in the runnable queue.
task_bound_ms_ticks	**bigint**	Point in time, in ms_ticks, when a task is bound to this worker.
worker_created_ms_ticks	**bigint**	Point in time, in ms_ticks, when a worker is created.
exception_num	**int**	Error number of the last exception that this worker encountered.
exception_severity	**int**	Severity of the last exception that this worker encountered.
exception_address	**varbinary(8)**	Code address that threw the exception
affinity	**bigint**	The thread affinity of the worker. Matches the affinity of the thread in sys.dm_os_threads (Transact-SQL).
state	**nvarchar(60)**	Worker state. Can be one of the following values: INIT = Worker is currently being initialized. RUNNING = Worker is currently running either nonpreemptively or preemptively. RUNNABLE = The worker is ready to run on the scheduler. SUSPENDED = The worker is currently suspended, waiting for an event to send it a signal.

COLUMN NAME	DATA TYPE	DESCRIPTION
start_quantum	**bigint**	Time, in milliseconds, at the start of the current run of this worker.
end_quantum	**bigint**	Time, in milliseconds, at the end of the current run of this worker.
last_wait_type	**nvarchar(60)**	Type of last wait. For a list of wait types, see sys.dm_os_wait_stats (Transact-SQL).
return_code	**int**	Return value from last wait. Can be one of the following values: 0 =SUCCESS 3 = DEADLOCK 4 = PREMATURE_WAKEUP 258 = TIMEOUT
quantum_used	**bigint**	Internal use only.
max_quantum	**bigint**	Internal use only.
boost_count	**int**	Internal use only.
tasks_processed_count	**int**	Number of tasks that this worker processed.
fiber_address	**varbinary(8)**	Memory address of the fiber with which this worker is associated. NULL = SQL Server is not configured for lightweight pooling.
task_address	**varbinary(8)**	Memory address of the current task. For more information, see sys.dm_os_tasks (Transact-SQL).
memory_object_address	**varbinary(8)**	Memory address of the worker memory object. For more information, see sys.dm_os_memory_objects (Transact-SQL).
thread_address	**varbinary(8)**	Memory address of the thread associated with this worker. For more information, see sys.dm_os_threads (Transact-SQL).
signal_worker_address	**varbinary(8)**	Memory address of the worker that last signaled this object. For more information, see sys.dm_os_workers.

COLUMN NAME	DATA TYPE	DESCRIPTION
scheduler_address	**varbinary(8)**	Memory address of the scheduler. For more information, see sys.dm_os_schedulers (Transact-SQL).
processor_group	**smallint**	Stores the processor group ID that is assigned to this thread.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Remarks

If the worker state is RUNNING and the worker is running nonpreemptively, the worker address matches the active_worker_address in sys.dm_os_schedulers.

When a worker that is waiting on an event is signaled, the worker is placed at the head of the runnable queue. SQL Server allows for this to happen one thousand times in a row, after which the worker is placed at the end of the queue. Moving a worker to the end of the queue has some performance implications.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

You can use the following query to find out how long a worker has been running in a SUSPENDED or RUNNABLE state.

```
SELECT
    t1.session_id,
    CONVERT(varchar(10), t1.status) AS status,
    CONVERT(varchar(15), t1.command) AS command,
    CONVERT(varchar(10), t2.state) AS worker_state,
    w_suspended =
      CASE t2.wait_started_ms_ticks
        WHEN 0 THEN 0
        ELSE
          t3.ms_ticks - t2.wait_started_ms_ticks
      END,
    w_runnable =
      CASE t2.wait_resumed_ms_ticks
        WHEN 0 THEN 0
        ELSE
          t3.ms_ticks - t2.wait_resumed_ms_ticks
      END
  FROM sys.dm_exec_requests AS t1
  INNER JOIN sys.dm_os_workers AS t2
    ON t2.task_address = t1.task_address
  CROSS JOIN sys.dm_os_sys_info AS t3
  WHERE t1.scheduler_id IS NOT NULL;
```

Here is the result set.

```
session_id status     command         worker_state w_suspended w_runnable
---------- ---------- --------------- ------------ ----------- --------------------
4          background LAZY WRITER     SUSPENDED    688         688
6          background LOCK MONITOR    SUSPENDED    4657        4657
19         background BRKR TASK       SUSPENDED    603820344   603820344
14         background BRKR EVENT HNDL SUSPENDED    63583641    63583641
51         running    SELECT          RUNNING      0           0
2          background RESOURCE MONITO RUNNING      0           603825954
3          background LAZY WRITER     SUSPENDED    422         422
7          background SIGNAL HANDLER  SUSPENDED    603820485   603820485
13         background TASK MANAGER    SUSPENDED    603824704   603824704
18         background BRKR TASK       SUSPENDED    603820407   603820407
9          background TRACE QUEUE TAS SUSPENDED    454         454
52         suspended  SELECT          SUSPENDED    35094       35094
1          background RESOURCE MONITO RUNNING      0           603825954
```

In the output, when `w_runnable` and `w_suspended` are equal, this represents the time that the worker is in the SUSPENDED state. Otherwise, `w_runnable` represents the time that is spent by the worker in the RUNNABLE state. In the output, session `52` is `SUSPENDED` for `35,094` milliseconds.

See Also

SQL Server Operating System Related Dynamic Management Views (Transact-SQL)

Stretch Database - sys.dm_db_rda_migration_status

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Contains one row for each batch of migrated data from each Stretch-enabled table on the local instance of SQL Server. Batches are identified by their start time and end time.

sys.dm_db_rda_migration_status is scoped to the current database context. Make sure you're in the database context of the Stretch-enable tables for which you want to see migration status.

In SQL Server 2016 (13.x), the output of **sys.dm_db_rda_migration_status** is limited to 200 rows.

COLUMN NAME	DATA TYPE	DESCRIPTION
table_id	**int**	The ID of the table from which rows were migrated.
database_id	**int**	The ID of the database from which rows were migrated.
migrated_rows	**bigint**	The number of rows migrated in this batch.
start_time_utc	**datetime**	The UTC time at which the batch started.
end_time_utc	**datetime**	The UTC time at which the batch finished.
error_number	**int**	If the batch fails, the error number of the error that occurred; otherwise, null.
error_severity	**int**	If the batch fails, the severity of the error that occurred; otherwise, null.
error_state	**int**	If the batch fails, the state of the error that occurred; otherwise, null. The **error_state** indicates the condition or location where the error occurred.

See Also

Stretch Database

Stretch Database - sys.dm_db_rda_schema_update_status

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Contains one row for each schema update task for the remote data archive of each Stretch-enabled table in the current database. Tasks are identified by their task ids.

dm_db_rda_schema_update_status is scoped to the current database context. Make sure you are in the database context of the Stretch-enabled table for which you want to see schema update status.

COLUMN NAME	DATA TYPE	DESCRIPTION
table_id	**int**	The ID of the local Stretch-enabled table whose remote data archive schema is being updated.
database_id	**int**	The ID of the database that contains the local Stretch-enabled table.
task_id	**bigint**	The ID of the remote data archive schema update task.
task_type	**int**	The type of the remote data archive schema update task.
task_type_desc	**nvarchar**	The description of the type of the remote data archive schema update task.
task_state	**int**	The state of the remote data archive schema update task.
task_state_des	**nvarchar**	The description of the state of the remote data archive schema update task.
start_time_utc	**datetime**	The UTC time at which the remote data archive schema update started.
end_time_utc	**datetime**	The UTC time at which the remote data archive schema update finished.
error_number	**int**	If the remote data archive schema update fails, the error number of the error that occurred; otherwise, null.
error_severity	**int**	If the remote data archive schema update fails, the severity of the error that occurred; otherwise, null.

COLUMN NAME	DATA TYPE	DESCRIPTION
error_state	**int**	If the remote data archive schema update fails, the state of the error that occurred; otherwise, null. The error_state indicates the condition or location where the error occurred.

See Also

Stretch Database

Transaction Related Dynamic Management Views and Functions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section contains the following dynamic management objects.

sys.dm_tran_active_snapshot_database_transactions (Transact-SQL)	sys.dm_tran_active_transactions (Transact-SQL)
sys.dm_tran_current_snapshot (Transact-SQL)	sys.dm_tran_current_transaction (Transact-SQL)
sys.dm_tran_database_transactions (Transact-SQL)	sys.dm_tran_locks (Transact-SQL)
sys.dm_tran_session_transactions (Transact-SQL)	sys.dm_tran_top_version_generators (Transact-SQL)
sys.dm_tran_transactions_snapshot (Transact-SQL)	sys.dm_tran_version_store (Transact-SQL)
sys.dm_tran_version_store_space_usage	

sys.dm_tran_active_snapshot_database_transactions (Transact-SQL)

5/4/2018 • 4 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In a SQL Server instance, this dynamic management view returns a virtual table for all active transactions that generate or potentially access row versions. Transactions are included for one or more of the following conditions:

- When either or both ALLOW_SNAPSHOT_ISOLATION and READ_COMMITTED_SNAPSHOT database options are set to ON:
  - There is one row for each transaction that is running under snapshot isolation level, or read-committed isolation level that is using row versioning.
  - There is one row for each transaction that causes a row version to be created in the current database. For example, the transaction generates a row version by updating or deleting a row in the current database.
- When a trigger is fired, there is one row for the transaction under which the trigger is executing.
- When an online indexing procedure is running, there is one row for the transaction that is creating the index.
- When Multiple Active Results Sets (MARS) session is enabled, there is one row for each transaction that is accessing row versions.

  This dynamic management view does not include system transactions.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_tran_active_snapshot_database_transactions**.

Syntax

```
sys.dm_tran_active_snapshot_database_transactions
```

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_id	**bigint**	Unique identification number assigned for the transaction. The transaction ID is primarily used to identify the transaction in locking operations.

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_sequence_num	**bigint**	Transaction sequence number. This is a unique sequence number that is assigned to a transaction when it starts. Transactions that do not generate version records and do not use snapshot scans will not receive a transaction sequence number.
commit_sequence_num	**bigint**	Sequence number that indicates when the transaction finishes (commits or stops). For active transactions, the value is NULL.
is_snapshot	**int**	0 = Is not a snapshot isolation transaction. 1 = Is a snapshot isolation transaction.
session_id	**int**	ID of the session that started the transaction.
first_snapshot_sequence_num	**bigint**	Lowest transaction sequence number of the transactions that were active when a snapshot was taken. On execution, a snapshot transaction takes a snapshot of all of the active transactions at that time. For nonsnapshot transactions, this column shows 0.
max_version_chain_traversed	**int**	Maximum length of the version chain that is traversed to find the transactionally consistent version.
average_version_chain_traversed	**real**	Average number of row versions in the version chains that are traversed.
elapsed_time_seconds	**bigint**	Elapsed time since the transaction obtained its transaction sequence number.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

sys.dm_tran_active_snapshot_database_transactions reports transactions that are assigned a transaction sequence number (XSN). The XSN is assigned when the transaction first accesses the version store. In a database that is enabled for snapshot isolation or read committed isolation using row versioning, the examples show when an XSN is assigned to a transaction:

- If a transaction is running under serializable isolation level, an XSN is assigned when the transaction first executes a statement, such as an UPDATE operation, that causes a row version to be created.
- If a transaction is running under snapshot isolation, an XSN is assigned when any data manipulation language (DML) statement, including a SELECT operation, is executed.

  Transaction sequence numbers are serially incremented for each transaction that is started in an instance of the Database Engine.

Examples

The following example uses a test scenario in which four concurrent transactions, each identified by a transaction sequence number (XSN), are running in a database that has the ALLOW_SNAPSHOT_ISOLATION and READ_COMMITTED_SNAPSHOT options set to ON. The following transactions are running:

- XSN-57 is an update operation under serializable isolation.
- XSN-58 is the same as XSN-57.
- XSN-59 is a select operation under snapshot isolation
- XSN-60 is same as XSN-59.

  The following query is executed.

```
SELECT
    transaction_id,
    transaction_sequence_num,
    commit_sequence_num,
    is_snapshot session_id,
    first_snapshot_sequence_num,
    max_version_chain_traversed,
    average_version_chain_traversed,
    elapsed_time_seconds
  FROM sys.dm_tran_active_snapshot_database_transactions;
```

Here is the result set.

```
transaction_id  transaction_sequence_num  commit_sequence_num
--------------  ------------------------  -------------------
9295            57                        NULL
9324            58                        NULL
9387            59                        NULL
9400            60                        NULL

is_snapshot  session_id   first_snapshot_sequence_num
-----------  -----------  ---------------------------
0            54           0
0            53           0
1            52           57
1            51           57

max_version_chain_traversed  average_version_chain_traversed
---------------------------  -------------------------------
0                            0
0                            0
1                            1
1                            1

elapsed_time_seconds
--------------------
419
397
359
333
```

The following information evaluates the results from **sys.dm_tran_active_snapshot_database_transactions**:

- XSN-57: Because this transaction is not running under snapshot isolation, the `is_snapshot` value and `first_snapshot_sequence_num` are `0`. `transaction_sequence_num` shows that a transaction sequence number has been assigned to this transaction, because one or both ALLOW_SNAPSHOT_ISOLATION or READ_COMMITTED_SNAPSHOT database options are ON.
- XSN-58: This transaction is not running under snapshot isolation and the same information for XSN-57 applies.
- XSN-59: This is the first active transaction that is running under snapshot isolation. This transaction reads data that is committed before XSN-57, as indicated by `first_snapshot_sequence_num`. The output for this transaction also shows the maximum version chain that is traversed for a row is `1` and has traversed an average of `1` version for each row that is accessed. This means that transactions XSN-57, XSN-58, and XSN-60 have not modified rows and committed.
- XSN-60: This is the second transaction running under snapshot isolation. The output shows the same information as XSN-59.

See Also

SET TRANSACTION ISOLATION LEVEL (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)
Transaction Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_tran_active_transactions (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about transactions for the instance of SQL Server.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_tran_active_transactions**.

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_id	**bigint**	ID of the transaction at the instance level, not the database level. It is only unique across all databases within an instance but not unique across all server instances.
name	**nvarchar(32)**	Transaction name. This is overwritten if the transaction is marked and the marked name replaces the transaction name.
transaction_begin_time	**datetime**	Time that the transaction started.
transaction_type	**int**	Type of transaction. 1 = Read/write transaction 2 = Read-only transaction 3 = System transaction 4 = Distributed transaction
transaction_uow	**uniqueidentifier**	Transaction unit of work (UOW) identifier for distributed transactions. MS DTC uses the UOW identifier to work with the distributed transaction.

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_state	**int**	0 = The transaction has not been completely initialized yet. 1 = The transaction has been initialized but has not started. 2 = The transaction is active. 3 = The transaction has ended. This is used for read-only transactions. 4 = The commit process has been initiated on the distributed transaction. This is for distributed transactions only. The distributed transaction is still active but further processing cannot take place. 5 = The transaction is in a prepared state and waiting resolution. 6 = The transaction has been committed. 7 = The transaction is being rolled back. 8 = The transaction has been rolled back.
transaction_status	**int**	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
transaction_status2	**int**	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
dtc_state	**int**	**Applies to**: Azure SQL Database (Initial release through current release). 1 = ACTIVE 2 = PREPARED 3 = COMMITTED 4 = ABORTED 5 = RECOVERED
dtc_status	**int**	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
dtc_isolation_level	**int**	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

COLUMN NAME	DATA TYPE	DESCRIPTION
filestream_transaction_id	**varbinary(128)**	**Applies to**: Azure SQL Database (Initial release through current release). Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

sys.dm_tran_session_transactions (Transact-SQL)
sys.dm_tran_database_transactions (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)
Transaction Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_tran_current_snapshot (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a virtual table that displays all active transactions at the time when the current snapshot transaction starts. If the current transaction is not a snapshot transaction, this function returns no rows.

sys.dm_tran_current_snapshot is similar to **sys.dm_tran_transactions_snapshot**, except that **sys.dm_tran_current_snapshot** returns only the active transactions for the current snapshot transaction.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_tran_current_snapshot**.

Syntax

```
sys.dm_tran_current_snapshot
```

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_sequence_num	**bigint**	Transaction sequence number of the active transaction.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example uses a test scenario in which four concurrent transactions, each identified by a transaction sequence number (XSN), are running in a database that has the ALLOW_SNAPSHOT_ISOLATION and READ_COMMITTED_SNAPSHOT options set to ON. The following transactions are running:

- XSN-57 is an update operation under serializable isolation.
- XSN-58 is the same as XSN-57.
- XSN-59 is a select operation under snapshot isolation.

- XSN-60 is the same as XSN-59.

  The following query is executed within the scope of XSN-59.

```
SELECT
    transaction_sequence_num
  FROM sys.dm_tran_current_snapshot;
```

Here is the result set.

```
transaction_sequence_num
------------------------
57
58
```

The results show that XSN-57 and XSN-58 were active at the time that the snapshot transaction XSN-59 started. This same result persists, even after XSN-57 and XSN-58 commit or roll back, until the snapshot transaction finishes.

The same query is executed within the scope of XSN-60.

Here is the result set.

```
transaction_sequence_num
------------------------
57
58
59
```

The output for XSN-60 includes the same transactions that appear for XSN-59, but also includes XSN-59, which was active when XSN-60 started.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Transaction Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_tran_current_transaction (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a single row that displays the state information of the transaction in the current session.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_tran_current_transaction**.

Syntax

```
sys.dm_tran_current_transaction
```

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_id	**bigint**	Transaction ID of the current snapshot.
transaction_sequence_num	**bigint**	Sequence number of the transaction that generates the record version.
transaction_is_snapshot	**bit**	Snapshot isolation state. This value is 1 if the transaction is started under snapshot isolation. Otherwise, the value is 0.
first_snapshot_sequence_num	**bigint**	Lowest transaction sequence number of the transactions that were active when a snapshot was taken. On execution, a snapshot transaction takes a snapshot of all of the active transactions at that time. For nonsnapshot transactions, this column shows 0.
last_transaction_sequence_num	**bigint**	Global sequence number. This value represents the last transaction sequence number that was generated by the system.

COLUMN NAME	DATA TYPE	DESCRIPTION
first_useful_sequence_num	**bigint**	Global sequence number. This value represents the oldest transaction sequence number of the transaction that has row versions that must be retained in the version store. Row versions that were created by prior transactions can be removed.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example uses a test scenario in which four concurrent transactions, each identified by a transaction sequence number (XSN), are running in a database that has the ALLOW_SNAPSHOT_ISOLATION and READ_COMMITTED_SNAPSHOT options set to ON. The following transactions are running:

- XSN-57 is an update operation under serializable isolation.
- XSN-58 is the same as XSN-57.
- XSN-59 is a select operation under snapshot isolation.
- XSN-60 is the same as XSN-59.

  The following query is executed within the scope of each transaction.

```
SELECT
    transaction_id
   ,transaction_sequence_num
   ,transaction_is_snapshot
   ,first_snapshot_sequence_num
   ,last_transaction_sequence_num
   ,first_useful_sequence_num
  FROM sys.dm_tran_current_transaction;
```

Here is the result for XSN-59.

```
transaction_id       transaction_sequence_num transaction_is_snapshot
-------------------- ------------------------ -----------------------
9387                 59                       1

first_snapshot_sequence_num last_transaction_sequence_num
--------------------------- -----------------------------
57                               61

first_useful_sequence_num
-------------------------
57
```

The output shows that XSN-59 is a snapshot transaction that uses XSN-57 as the first transaction that was active when XSN-59 started. This means that XSN-59 reads data committed by transactions that have a transaction sequence number lower than XSN-57.

Here is the result for XSN-57.

```
transaction_id       transaction_sequence_num transaction_is_snapshot
-------------------- ------------------------ -----------------------
9295                 57                       0

first_snapshot_sequence_num last_transaction_sequence_num
--------------------------- -----------------------------
NULL                        61

first_useful_sequence_num
-------------------------
57
```

Because XSN-57 is not a snapshot transaction, `first_snapshot_sequence_num` is `NULL`.

See Also

[Dynamic Management Views and Functions (Transact-SQL)](#)
[Transaction Related Dynamic Management Views and Functions (Transact-SQL)](#)

sys.dm_tran_database_transactions (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about transactions at the database level.

> **NOTE**
> To call this DMV from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_tran_database_transactions**.

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_id	**bigint**	ID of the transaction at the instance level, not the database level. It is only unique across all databases within an instance, but not unique across all server instances.
database_id	**int**	ID of the database associated with the transaction.
database_transaction_begin_time	**datetime**	Time at which the database became involved in the transaction. Specifically, it is the time of the first log record in the database for the transaction.
database_transaction_type	**int**	1 = Read/write transaction 2 = Read-only transaction 3 = System transaction

COLUMN NAME	DATA TYPE	DESCRIPTION
database_transaction_state	**int**	1 = The transaction has not been initialized. 3 = The transaction has been initialized but has not generated any log records. 4 = The transaction has generated log records. 5 = The transaction has been prepared. 10 = The transaction has been committed. 11 = The transaction has been rolled back. 12 = The transaction is being committed. (The log record is being generated, but has not been materialized or persisted.)
database_transaction_status	**int**	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
database_transaction_status2	**int**	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
database_transaction_log_record_count	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2017. Number of log records generated in the database for the transaction.
database_transaction_replicate_record_count	**int**	**Applies to**: SQL Server 2008 through SQL Server 2017. Number of log records generated in the database for the transaction that is replicated.
database_transaction_log_bytes_used	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2017. Number of bytes used so far in the database log for the transaction.
database_transaction_log_bytes_reserved	**bigint**	**Applies to**: SQL Server 2008 through SQL Server 2017. Number of bytes reserved for use in the database log for the transaction.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_transaction_log_bytes_used_system	**int**	**Applies to**: SQL Server 2008 through SQL Server 2017. Number of bytes used so far in the database log for system transactions on behalf of the transaction.
database_transaction_log_bytes_reserved_system	**int**	**Applies to**: SQL Server 2008 through SQL Server 2017. Number of bytes reserved for use in the database log for system transactions on behalf of the transaction.
database_transaction_begin_lsn	**numeric(25,0)**	**Applies to**: SQL Server 2008 through SQL Server 2017. Log sequence number (LSN) of the begin record for the transaction in the database log.
database_transaction_last_lsn	**numeric(25,0)**	**Applies to**: SQL Server 2008 through SQL Server 2017. LSN of the most recently logged record for the transaction in the database log.
database_transaction_most_recent_savepoint_lsn	**numeric(25,0)**	**Applies to**: SQL Server 2008 through SQL Server 2017. LSN of the most recent savepoint for the transaction in the database log.
database_transaction_commit_lsn	**numeric(25,0)**	**Applies to**: SQL Server 2008 through SQL Server 2017. LSN of the commit log record for the transaction in the database log.
database_transaction_last_rollback_lsn	**numeric(25,0)**	**Applies to**: SQL Server 2008 through SQL Server 2017. LSN that was most recently rolled back to. If no rollback has taken place, the value is MaxLSN.
database_transaction_next_undo_lsn	**numeric(25,0)**	**Applies to**: SQL Server 2008 through SQL Server 2017. LSN of the next record to undo.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

See Also

sys.dm_tran_active_transactions (Transact-SQL)
sys.dm_tran_session_transactions (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)
Transaction Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_tran_locks (Transact-SQL)

5/4/2018 • 21 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about currently active lock manager resources in SQL Server 2017. Each row represents a currently active request to the lock manager for a lock that has been granted or is waiting to be granted.

The columns in the result set are divided into two main groups: resource and request. The resource group describes the resource on which the lock request is being made, and the request group describes the lock request.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_tran_locks**.

COLUMN NAME	DATA TYPE	DESCRIPTION
resource_type	**nvarchar(60)**	Represents the resource type. The value can be one of the following: DATABASE, FILE, OBJECT, PAGE, KEY, EXTENT, RID, APPLICATION, METADATA, HOBT, or ALLOCATION_UNIT.
resource_subtype	**nvarchar(60)**	Represents a subtype of **resource_type**. Acquiring a subtype lock without holding a nonsubtyped lock of the parent type is technically valid. Different subtypes do not conflict with each other or with the nonsubtyped parent type. Not all resource types have subtypes.
resource_database_id	**int**	ID of the database under which this resource is scoped. All resources handled by the lock manager are scoped by the database ID.
resource_description	**nvarchar(256)**	Description of the resource that contains only information that is not available from other resource columns.
resource_associated_entity_id	**bigint**	ID of the entity in a database with which a resource is associated. This can be an object ID, Hobt ID, or an Allocation Unit ID, depending on the resource type.

COLUMN NAME	DATA TYPE	DESCRIPTION
resource_lock_partition	**Int**	ID of the lock partition for a partitioned lock resource. The value for nonpartitioned lock resources is 0.
request_mode	**nvarchar(60)**	Mode of the request. For granted requests, this is the granted mode; for waiting requests, this is the mode being requested.
request_type	**nvarchar(60)**	Request type. The value is LOCK.
request_status	**nvarchar(60)**	Current status of this request. Possible values are GRANTED, CONVERT, WAIT, LOW_PRIORITY_CONVERT, LOW_PRIORITY_WAIT, or ABORT_BLOCKERS. For more information about low priority waits and abort blockers, see the *low_priority_lock_wait* section of ALTER INDEX (Transact-SQL).
request_reference_count	**smallint**	Returns an approximate number of times the same requestor has requested this resource.
request_lifetime	**int**	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
request_session_id	**int**	Session ID that currently owns this request. The owning session ID can change for distributed and bound transactions. A value of -2 indicates that the request belongs to an orphaned distributed transaction. A value of -3 indicates that the request belongs to a deferred recovery transaction, such as, a transaction for which a rollback has been deferred at recovery because the rollback could not be completed successfully.
request_exec_context_id	**int**	Execution context ID of the process that currently owns this request.
request_request_id	**int**	Request ID (batch ID) of the process that currently owns this request. This value will change every time that the active Multiple Active Result Set (MARS) connection for a transaction changes.

COLUMN NAME	DATA TYPE	DESCRIPTION
request_owner_type	**nvarchar(60)**	Entity type that owns the request. Lock manager requests can be owned by a variety of entities. Possible values are: TRANSACTION = The request is owned by a transaction. CURSOR = The request is owned by a cursor. SESSION = The request is owned by a user session. SHARED_TRANSACTION_WORKSPACE = The request is owned by the shared part of the transaction workspace. EXCLUSIVE_TRANSACTION_WORKSPACE = The request is owned by the exclusive part of the transaction workspace. NOTIFICATION_OBJECT = The request is owned by an internal SQL Server component. This component has requested the lock manager to notify it when another component is waiting to take the lock. The FileTable feature is a component that uses this value. **Note:** Work spaces are used internally to hold locks for enlisted sessions.
request_owner_id	**bigint**	ID of the specific owner of this request. When a transaction is the owner of the request, this value contains the transaction ID. When a FileTable is the owner of the request, **request_owner_id** has one of the following values. -4 : A FileTable has taken a database lock. -3 : A FileTable has taken a table lock. Other value : The value represents a file handle. This value also appears as **fcb_id** in the dynamic management view sys.dm_filestream_non_transacted_handles (Transact-SQL).

COLUMN NAME	DATA TYPE	DESCRIPTION
request_owner_guid	**uniqueidentifier**	GUID of the specific owner of this request. This value is only used by a distributed transaction where the value corresponds to the MS DTC GUID for that transaction.
request_owner_lockspace_id	**nvarchar(32)**	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. This value represents the lockspace ID of the requestor. The lockspace ID determines whether two requestors are compatible with each other and can be granted locks in modes that would otherwise conflict with one another.
lock_owner_address	**varbinary(8)**	Memory address of the internal data structure that is used to track this request. This column can be joined the with **resource_address** column in **sys.dm_os_waiting_tasks**.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

A granted request status indicates that a lock has been granted on a resource to the requestor. A waiting request indicates that the request has not yet been granted. The following waiting-request types are returned by the **request_status** column:

- A convert request status indicates that the requestor has already been granted a request for the resource and is currently waiting for an upgrade to the initial request to be granted.

- A wait request status indicates that the requestor does not currently hold a granted request on the resource.

  Because **sys.dm_tran_locks** is populated from internal lock manager data structures, maintaining this information does not add extra overhead to regular processing. Materializing the view does require access to the lock manager internal data structures. This can have minor effects on the regular processing in the server. These effects should be unnoticeable and should only affect heavily used resources. Because the data in this view corresponds to live lock manager state, the data can change at any time, and rows are added and removed as locks are acquired and released. This view has no historical information.

Two requests operate on the same resource only if all the resource-group columns are equal.

You can control the locking of read operations by using the following tools:

- SET TRANSACTION ISOLATION LEVEL to specify the level of locking for a session. For more information, see SET TRANSACTION ISOLATION LEVEL (Transact-SQL).

- Locking table hints to specify the level of locking for an individual reference of a table in a FROM clause. For syntax and restrictions, see Table Hints (Transact-SQL).

  A resource that is running under one session ID can have more than one granted lock. Different entities that are running under one session can each own a lock on the same resource, and the information is displayed in the **request_owner_type** and **request_owner_id** columns that are returned by **sys.dm_tran_locks**. If multiple instances of the same **request_owner_type** exist, the **request_owner_id** column is used to distinguish each instance. For distributed transactions, the **request_owner_type** and the **request_owner_guid** columns will show the different entity information.

  For example, Session S1 owns a shared lock on **Table1**; and transaction T1, which is running under session S1, also owns a shared lock on **Table1**. In this case, the **resource_description** column that is returned by **sys.dm_tran_locks** will show two instances of the same resource. The **request_owner_type** column will show one instance as a session and the other as a transaction. Also, the **resource_owner_id** column will have different values.

  Multiple cursors that run under one session are indistinguishable and are treated as one entity.

  Distributed transactions that are not associated with a session ID value are orphaned transactions and are assigned the session ID value of -2. For more information, see KILL (Transact-SQL).

Resource Details

The following table lists the resources that are represented in the **resource_associated_entity_id** column.

RESOURCE TYPE	RESOURCE DESCRIPTION	RESOURCE_ASSOCIATED_ENTITY_ID
DATABASE	Represents a database.	Not applicable
FILE	Represents a database file. This file can be either a data or a log file.	Not applicable
OBJECT	Represents a database object. This object can be a data table, view, stored procedure, extended stored procedure, or any object that has an object ID.	Object ID
PAGE	Represents a single page in a data file.	HoBt ID. This value corresponds to **sys.partitions.hobt_id**. The HoBt ID is not always available for PAGE resources because the HoBt ID is extra information that can be provided by the caller, and not all callers can provide this information.
KEY	Represents a row in an index.	HoBt ID. This value corresponds to **sys.partitions.hobt_id**.

RESOURCE TYPE	RESOURCE DESCRIPTION	RESOURCE_ASSOCIATED_ENTITY_ID
EXTENT	Represents a data file extent. An extent is a group of eight contiguous pages.	Not applicable
RID	Represents a physical row in a heap.	HoBt ID. This value corresponds to **sys.partitions.hobt_id**. The HoBt ID is not always available for RID resources because the HoBt ID is extra information that can be provided by the caller, and not all callers can provide this information.
APPLICATION	Represents an application specified resource.	Not applicable
METADATA	Represents metadata information.	Not applicable
HOBT	Represents a heap or a B-tree. These are the basic access path structures.	HoBt ID. This value corresponds to **sys.partitions.hobt_id**.
ALLOCATION_UNIT	Represents a set of related pages, such as an index partition. Each allocation unit covers a single Index Allocation Map (IAM) chain.	Allocation Unit ID. This value corresponds to **sys.allocation_units.allocation_unit_id**.

The following table lists the subtypes that are associated with each resource type.

RESOURCESUBTYPE	SYNCHRONIZES
ALLOCATION_UNIT.BULK_OPERATION_PAGE	Pre-allocated pages used for bulk operations.
ALLOCATION_UNIT.PAGE_COUNT	Allocation unit page count statistics during deferred drop operations.
DATABASE.BULKOP_BACKUP_DB	Database backups with bulk operations.
DATABASE.BULKOP_BACKUP_LOG	Database log backups with bulk operations.
DATABASE.CHANGE_TRACKING_CLEANUP	Change tracking cleanup tasks.
DATABASE.CT_DDL	Database and table-level change tracking DDL operations.
DATABASE.CONVERSATION_PRIORITY	Service Broker conversation priority operations such as CREATE BROKER PRIORITY.
DATABASE.DDL	Data definition language (DDL) operations with filegroup operations, such as drop.
DATABASE.ENCRYPTION_SCAN	TDE encryption synchronization.
DATABASE.PLANGUIDE	Plan guide synchronization.

RESOURCESUBTYPE	SYNCHRONIZES
DATABASE.RESOURCE_GOVERNOR_DDL	DDL operations for resource governor operations such as ALTER RESOURCE POOL.
DATABASE.SHRINK	Database shrink operations.
DATABASE.STARTUP	Used for database startup synchronization.
FILE.SHRINK	File shrink operations.
HOBT.BULK_OPERATION	Heap-optimized bulk load operations with concurrent scan, under these isolation levels: snapshot, read uncommitted, and read committed using row versioning.
HOBT.INDEX_REORGANIZE	Heap or index reorganization operations.
OBJECT.COMPILE	Stored procedure compile.
OBJECT.INDEX_OPERATION	Index operations.
OBJECT.UPDSTATS	Statistics updates on a table.
METADATA.ASSEMBLY	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ASSEMBLY_CLR_NAME	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ASSEMBLY_TOKEN	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ASYMMETRIC_KEY	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.AUDIT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.AUDIT_ACTIONS	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.AUDIT_SPECIFICATION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.AVAILABILITY_GROUP	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CERTIFICATE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CHILD_INSTANCE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.COMPRESSED_FRAGMENT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

RESOURCESUBTYPE	SYNCHRONIZES
METADATA.COMPRESSED_ROWSET	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CONVERSTATION_ENDPOINT_RECV	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CONVERSTATION_ENDPOINT_SEND	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CONVERSATION_GROUP	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CONVERSATION_PRIORITY	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CREDENTIAL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CRYPTOGRAPHIC_PROVIDER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DATA_SPACE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DATABASE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DATABASE_PRINCIPAL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DB_MIRRORING_SESSION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DB_MIRRORING_WITNESS	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DB_PRINCIPAL_SID	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ENDPOINT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ENDPOINT_WEBMETHOD	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.EXPR_COLUMN	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.EXPR_HASH	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.FULLTEXT_CATALOG	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

RESOURCESUBTYPE	SYNCHRONIZES
METADATA.FULLTEXT_INDEX	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.FULLTEXT_STOPLIST	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.INDEX_EXTENSION_SCHEME	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.INDEXSTATS	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.INSTANTIATED_TYPE_HASH	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.MESSAGE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.METADATA_CACHE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PARTITION_FUNCTION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PASSWORD_POLICY	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PERMISSIONS	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PLAN_GUIDE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PLAN_GUIDE_HASH	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PLAN_GUIDE_SCOPE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.QNAME	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.QNAME_HASH	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.REMOTE_SERVICE_BINDING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ROUTE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SCHEMA	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

RESOURCESUBTYPE	SYNCHRONIZES
METADATA.SECURITY_CACHE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SECURITY_DESCRIPTOR	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SEQUENCE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVER_EVENT_SESSIONS	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVER_PRINCIPAL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVICE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVICE_BROKER_GUID	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVICE_CONTRACT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVICE_MESSAGE_TYPE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.STATS	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SYMMETRIC_KEY	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.USER_TYPE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.XML_COLLECTION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.XML_COMPONENT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.XML_INDEX_QNAME	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

The following table provides the format of the **resource_description** column for each resource type.

RESOURCE	FORMAT	DESCRIPTION
DATABASE	Not applicable	Database ID is already available in the **resource_database_id** column.

RESOURCE	FORMAT	DESCRIPTION
FILE	<file_id>	ID of the file that is represented by this resource.
OBJECT	<object_id>	ID of the object that is represented by this resource. This object can be any object listed in **sys.objects**, not just a table.
PAGE	<file_id>:<page_in_file>	Represents the file and page ID of the page that is represented by this resource.
KEY	<hash_value>	Represents a hash of the key columns from the row that is represented by this resource.
EXTENT	<file_id>:<page_in_files>	Represents the file and page ID of the extent that is represented by this resource. The extent ID is the same as the page ID of the first page in the extent.
RID	<file_id>:<page_in_file>:<row_on_page>	Represents the page ID and row ID of the row that is represented by this resource. Note that if the associated object ID is 99, this resource represents one of the eight mixed page slots on the first IAM page of an IAM chain.
APPLICATION	<DbPrincipalId>:<upto 32 characters>:(<hash_value>)	Represents the ID of the database principal that is used for scoping this application lock resource. Also included are up to 32 characters from the resource string that corresponds to this application lock resource. In certain cases, only 2 characters can be displayed due to the full string no longer being available. This behavior occurs only at database recovery time for application locks that are reacquired as part of the recovery process. The hash value represents a hash of the full resource string that corresponds to this application lock resource.
HOBT	Not applicable	HoBt ID is included as the **resource_associated_entity_id**.
ALLOCATION_UNIT	Not applicable	Allocation Unit ID is included as the **resource_associated_entity_id**.
METADATA.ASSEMBLY	assembly_id = A	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

RESOURCE	FORMAT	DESCRIPTION
METADATA.ASSEMBLY_CLR_NAME	$qname_id = Q	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ASSEMBLY_TOKEN	assembly_id = A, $token_id	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ASSYMMETRIC_KEY	asymmetric_key_id = A	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.AUDIT	audit_id = A	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.AUDIT_ACTIONS	device_id = D, major_id = M	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.AUDIT_SPECIFICATION	audit_specification_id = A	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.AVAILABILITY_GROUP	availability_group_id = A	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CERTIFICATE	certificate_id = C	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CHILD_INSTANCE	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.COMPRESSED_FRAGMENT	object_id = O , compressed_fragment_id = C	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.COMPRESSED_ROW	object_id = O	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CONVERSTATION_ENDPOINT_RECV	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CONVERSTATION_ENDPOINT_SEND	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CONVERSATION_GROUP	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

RESOURCE	FORMAT	DESCRIPTION
METADATA.CONVERSATION_PRIORITY	conversation_priority_id = C	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CREDENTIAL	credential_id = C	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.CRYPTOGRAPHIC_PROVIDER	provider_id = P	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DATA_SPACE	data_space_id = D	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DATABASE	database_id = D	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DATABASE_PRINCIPAL	principal_id = P	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DB_MIRRORING_SESSION	database_id = D	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DB_MIRRORING_WITNESS	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.DB_PRINCIPAL_SID	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ENDPOINT	endpoint_id = E	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ENDPOINT_WEBMETHOD	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.FULLTEXT_CATALOG	fulltext_catalog_id = F	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.FULLTEXT_INDEX	object_id = O	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.EXPR_COLUMN	object_id = O, column_id = C	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

RESOURCE	FORMAT	DESCRIPTION
METADATA.EXPR_HASH	object_id = O, $hash = H	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.FULLTEXT_CATALOG	fulltext_catalog_id = F	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.FULLTEXT_INDEX	object_id = O	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.FULLTEXT_STOPLIST	fulltext_stoplist_id = F	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.INDEX_EXTENSION_SCHEME	index_extension_id = I	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.INDEXSTATS	object_id = O, index_id or stats_id = I	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.INSTANTIATED_TYPE_HASH	user_type_id = U, hash = H	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.MESSAGE	message_id = M	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.METADATA_CACHE	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PARTITION_FUNCTION	function_id = F	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PASSWORD_POLICY	principal_id = P	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PERMISSIONS	class = C	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.PLAN_GUIDE	plan_guide_id = P	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA. PLAN_GUIDE_HASH	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

RESOURCE	FORMAT	DESCRIPTION
METADATA. PLAN_GUIDE_SCOPE	scope_id = S	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.QNAME	$qname_id = Q	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.QNAME_HASH	$qname_scope_id = Q, $qname_hash = H	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.REMOTE_SERVICE_BINDING	remote_service_binding_id = R	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.ROUTE	route_id = R	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SCHEMA	schema_id = S	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SECURITY_CACHE	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SECURITY_DESCRIPTOR	sd_id = S	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SEQUENCE	$seq_type = S, object_id = O	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVER	server_id = S	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVER_EVENT_SESSIONS	event_session_id = E	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVER_PRINCIPAL	principal_id = P	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVICE	service_id = S	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVICE_BROKER_GUID	$hash = H1:H2:H3	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

RESOURCE	FORMAT	DESCRIPTION
METADATA.SERVICE_CONTRACT	service_contract_id = S	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SERVICE_MESSAGE_TYPE	message_type_id = M	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.STATS	object_id = O, stats_id = S	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.SYMMETRIC_KEY	symmetric_key_id = S	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.USER_TYPE	user_type_id = U	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.XML_COLLECTION	xml_collection_id = X	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.XML_COMPONENT	xml_component_id = X	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.
METADATA.XML_INDEX_QNAME	object_id = O, $qname_id = Q	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.

The following XEvents are related to partition **SWITCH** and online index rebuild. For information about syntax, see ALTER TABLE (Transact-SQL) and ALTER INDEX (Transact-SQL).

- lock_request_priority_state
- process_killed_by_abort_blockers
- ddl_with_wait_at_low_priority

  The existing XEvent **progress_report_online_index_operation** for online index operations was extended by adding **partition_number** and **partition_id**.

Examples

A. Using sys.dm_tran_locks with other tools

The following example works with a scenario in which an update operation is blocked by another transaction. By using **sys.dm_tran_locks** and other tools, information about locking resources is provided.

```
USE tempdb;
GO

-- Create test table and index.
CREATE TABLE t_lock
    (
    c1 int, c2 int
    );
GO

CREATE INDEX t_lock_ci on t_lock(c1);
GO

-- Insert values into test table
INSERT INTO t_lock VALUES (1, 1);
INSERT INTO t_lock VALUES (2,2);
INSERT INTO t_lock VALUES (3,3);
INSERT INTO t_lock VALUES (4,4);
INSERT INTO t_lock VALUES (5,5);
INSERT INTO t_lock VALUES (6,6);
GO

-- Session 1
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRAN
    SELECT c1
        FROM t_lock
        WITH(holdlock, rowlock);

-- Session 2
BEGIN TRAN
    UPDATE t_lock SET c1 = 10
```

The following query will display lock information. The value for `<dbid>` should be replaced with the **database_id** from **sys.databases**.

```
SELECT resource_type, resource_associated_entity_id,
    request_status, request_mode,request_session_id,
    resource_description
    FROM sys.dm_tran_locks
    WHERE resource_database_id = <dbid>
```

The following query returns object information by using `resource_associated_entity_id` from the previous query. This query must be executed while you are connected to the database that contains the object.

```
SELECT object_name(object_id), *
    FROM sys.partitions
    WHERE hobt_id=<resource_associated_entity_id>
```

The following query will show blocking information.

```
SELECT
        t1.resource_type,
        t1.resource_database_id,
        t1.resource_associated_entity_id,
        t1.request_mode,
        t1.request_session_id,
        t2.blocking_session_id
    FROM sys.dm_tran_locks as t1
    INNER JOIN sys.dm_os_waiting_tasks as t2
        ON t1.lock_owner_address = t2.resource_address;
```

Release the resources by rolling back the transactions.

```
-- Session 1
ROLLBACK;
GO

-- Session 2
ROLLBACK;
GO
```

B. Linking session information to operating system threads

The following example returns information that associates a session ID with a Windows thread ID. The performance of the thread can be monitored in the Windows Performance Monitor. This query does not return session IDs that are currently sleeping.

```
SELECT STasks.session_id, SThreads.os_thread_id
    FROM sys.dm_os_tasks AS STasks
    INNER JOIN sys.dm_os_threads AS SThreads
        ON STasks.worker_address = SThreads.worker_address
    WHERE STasks.session_id IS NOT NULL
    ORDER BY STasks.session_id;
GO
```

See Also

sys.dm_tran_database_transactions (Transact-SQL)
Dynamic Management Views and Functions (Transact-SQL)
Transaction Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_tran_session_transactions (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns correlation information for associated transactions and sessions.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_tran_session_transactions**.

COLUMN NAME	DATA TYPE	DESCRIPTION
session_id	**int**	ID of the session under which the transaction is running.
transaction_id	**bigint**	ID of the transaction.
transaction_descriptor	**binary(8)**	Transaction identifier used by SQL Server when communicating with the client driver.
enlist_count	**int**	Number of active requests in the session working on the transaction.
is_user_transaction	**bit**	1 = The transaction was initiated by a user request. 0 = System transaction.
is_local	**bit**	1 = Local transaction. 0 = Distributed transaction or an enlisted bound session transaction.
is_enlisted	**bit**	1 = Enlisted distributed transaction. 0 = Not an enlisted distributed transaction.
is_bound	**bit**	1 = The transaction is active on the session via bound sessions. 0 = The transaction is not active on the session via bound sessions.
open_transaction_count		The number of open transactions for each session.

COLUMN NAME	DATA TYPE	DESCRIPTION
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

Through bound sessions and distributed transactions, it is possible for a transaction to be running under more than one session. In such cases, sys.dm_tran_session_transactions will show multiple rows for the same transaction_id, one for each session under which the transaction is running.

By executing multiple requests in autocommit mode using multiple active result sets (MARS), it is possible to have more than one active transaction on a single session. In such cases, sys.dm_tran_session_transactions will show multiple rows for the same session_id, one for each transaction running under that session.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Transaction Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_tran_top_version_generators (Transact-SQL)

5/4/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a virtual table for the objects that are producing the most versions in the version store. **sys.dm_tran_top_version_generators** returns the top 256 aggregated record lengths that are grouped by the **database_id** and **rowset_id**. **sys.dm_tran_top_version_generators** retrieves data by querying the **dm_tran_version_store** virtual table. **sys.dm_tran_top_version_generators** is an inefficient view to run because this view queries the version store, and the version store can be very large. We recommend that you use this function to find the largest consumers of the version store.

> **NOTE**
> To call this from Azure SQL Data Warehouse or Parallel Data Warehouse, use the name **sys.dm_pdw_nodes_tran_top_version_generators**.

Syntax

```
sys.dm_tran_top_version_generators
```

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	Database ID.
rowset_id	**bigint**	Rowset ID.
aggregated_record_length_in_bytes	**int**	Sum of the record lengths for each **database_id** and **rowset_id pair** in the version store.
pdw_node_id	**int**	**Applies to**: Azure SQL Data Warehouse, Parallel Data Warehouse The identifier for the node that this distribution is on.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

Because **sys.dm_tran_top_version_generators** might have to read many pages as it scans the entire version store, running **sys.dm_tran_top_version_generators** can interfere with system performance.

Examples

The following example uses a test scenario in which four concurrent transactions, each identified by a transaction sequence number (XSN), are running in a database that has the ALLOW_SNAPSHOT_ISOLATION and READ_COMMITTED_SNAPSHOT options set to ON. The following transactions are running:

- XSN-57 is an update operation under serializable isolation.
- XSN-58 is the same as XSN-57.
- XSN-59 is a select operation under snapshot isolation.
- XSN-60 is the same as XSN-59.

  The following query is executed.

```
SELECT
    database_id,
    rowset_id,
    aggregated_record_length_in_bytes
  FROM sys.dm_tran_top_version_generators;
```

Here is the result set.

```
database_id rowset_id            aggregated_record_length_in_bytes
----------- -------------------- --------------------------------
9           72057594038321152    87
9           72057594038386688    33
```

The output shows that all versions are created by `database_id``9` and that the versions generate from two tables.

See Also

[Dynamic Management Views and Functions (Transact-SQL)]
[Transaction Related Dynamic Management Views and Functions (Transact-SQL)]

sys.dm_tran_transactions_snapshot (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a virtual table for the **sequence_number** of transactions that are active when each snapshot transaction starts. The information that is returned by this view can you help you do the following:

- Find the number of currently active snapshot transactions.
- Identify data modifications that are ignored by a particular snapshot transaction. For a transaction that is active when a snapshot transaction starts, all data modifications by that transaction, even after that transaction commits, are ignored by the snapshot transaction.

  For example, consider the following output from **sys.dm_tran_transactions_snapshot**:

```
transaction_sequence_num snapshot_id snapshot_sequence_num
------------------------ ----------- ---------------------
59                       0           57
59                       0           58
60                       0           57
60                       0           58
60                       0           59
60                       3           57
60                       3           58
60                       3           59
60                       3           60
```

The `transaction_sequence_num` column identifies the transaction sequence (XSN) number of the current snapshot transactions. The output shows two: `59` and `60`. The `snapshot_sequence_num` column identifies the transaction sequence number of the transactions that are active when each snapshot transaction starts.

The output shows that snapshot transaction XSN-59 starts while two active transactions, XSN-57 and XSN-58, are running. If XSN-57 or XSN-58 makes data modifications, XSN-59 ignores the changes and uses row versioning to maintain a transactionally consistent view of the database.

Snapshot transaction XSN-60 ignores data modifications made by XSN-57 and XSN-58 and also XSN 59.

Syntax

```
dm_tran_transactions_snapshot
```

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_sequence_num	**bigint**	Transaction sequence number (XSN) of a snapshot transaction.

COLUMN NAME	DATA TYPE	DESCRIPTION
snapshot_id	**int**	Snapshot ID for each Transact-SQL statement started under read-committed using row versioning. This value is used to generate a transactionally consistent view of the database supporting each query that is being run under read-committed using row versioning.
snapshot_sequence_num	**bigint**	Transaction sequence number of a transaction that was active when the snapshot transaction started.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Remarks

When a snapshot transaction starts, the Database Engine records all of the transactions that are active at that time. **sys.dm_tran_transactions_snapshot** reports this information for all currently active snapshot transactions.

Each transaction is identified by a transaction sequence number that is assigned when the transaction begins. Transactions start at the time a BEGIN TRANSACTION or BEGIN WORK statement is executed. However, the Database Engine assigns the transaction sequence number with the execution of the first Transact-SQL statement that accesses data after the BEGIN TRANSACTION or BEGIN WORK statement. The transaction sequence numbers are incremented by one.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Transaction Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_tran_version_store (Transact-SQL)

5/4/2018 • 2 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a virtual table that displays all version records in the version store. **sys.dm_tran_version_store** is inefficient to run because it queries the entire version store, and the version store can be very large.

Each versioned record is stored as binary data together with some tracking or status information. Similar to records in database tables, version-store records are stored in 8192-byte pages. If a record exceeds 8192 bytes, the record will be split across two different records.

Because the versioned record is stored as binary, there are no problems with different collations from different databases. Use **sys.dm_tran_version_store** to find the previous versions of the rows in binary representation as they exist in the version store.

Syntax

```
sys.dm_tran_version_store
```

Table Returned

COLUMN NAME	DATA TYPE	DESCRIPTION
transaction_sequence_num	**bigint**	Sequence number of the transaction that generates the record version.
version_sequence_num	**bigint**	Version record sequence number. This value is unique within the version-generating transaction.
database_id	**int**	Database ID of the versioned record.
rowset_id	**bigint**	Rowset ID of the record.
status	**tinyint**	Indicates whether a versioned record has been split across two records. If the value is 0, the record is stored in one page. If the value is 1, the record is split into two records that are stored on two different pages.
min_length_in_bytes	**smallint**	Minimum length of the record in bytes.
record_length_first_part_in_bytes	**smallint**	Length of the first part of the versioned record in bytes.
record_image_first_part	**varbinary(8000)**	Binary image of the first part of version record.

COLUMN NAME	DATA TYPE	DESCRIPTION
record_length_second_part_in_bytes	**smallint**	Length of the second part of version record in bytes.
record_image_second_part	**varbinary(8000)**	Binary image of the second part of the version record.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.
On SQL Database, requires the `VIEW DATABASE STATE` permission in the database.

Examples

The following example uses a test scenario in which four concurrent transactions, each identified by a transaction sequence number (XSN), are running in a database that has the ALLOW_SNAPSHOT_ISOLATION and READ_COMMITTED_SNAPSHOT options set to ON. The following transactions are running:

- XSN-57 is an update operation under serializable isolation.
- XSN-58 is the same as XSN-57.
- XSN-59 is a select operation under snapshot isolation.
- XSN-60 is the same as XSN-59.

  The following query is executed.

```
SELECT
    transaction_sequence_num,
    version_sequence_num,
    database_id rowset_id,
    status,
    min_length_in_bytes,
    record_length_first_part_in_bytes,
    record_image_first_part,
    record_length_second_part_in_bytes,
    record_image_second_part
  FROM sys.dm_tran_version_store;
```

Here is the result set.

```
transaction_sequence_num version_sequence_num database_id
------------------------ -------------------- -----------
57                        1                    9
57                        2                    9
57                        3                    9
58                        1                    9

rowset_id            status min_length_in_bytes
-------------------- ------ -------------------
72057594038321152    0      12
72057594038321152    0      12
72057594038321152    0      12
72057594038386688    0      16

record_length_first_part_in_bytes
--------------------------------
29
29
29
33

record_image_first_part
--------------------------------------------------------------------
0x50000C0073000000010000000200FCB000000001000000270000000000
0x50000C0073000000020000000200FCB000000001000100270000000000
0x50000C0073000000030000000200FCB000000001000200270000000000
0x500010000100000002000000030000000300F800000000000000002E0000000000

record_length_second_part_in_bytes record_image_second_part
---------------------------------- -----------------------
0                                  NULL
0                                  NULL
0                                  NULL
0                                  NULL
```

The output shows that XSN-57 has created three row versions from one table and XSN-58 has created one row version from another table.

See Also

Dynamic Management Views and Functions (Transact-SQL)
Transaction Related Dynamic Management Views and Functions (Transact-SQL)

sys.dm_tran_version_store_space_usage (Transact-SQL)

5/3/2018 • 1 min to read • Edit Online

THIS TOPIC APPLIES TO: SQL Server (starting with 2016 SP2) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a table that displays total space in tempdb used by version store records for each database. **sys.dm_tran_version_store_space_usage** is efficient and not expensive to run, as it does not navigate through individual version store records, and returns aggregated version store space consumed in tempdb per database.

Each versioned record is stored as binary data, together with some tracking or status information. Similar to records in database tables, version-store records are stored in 8192-byte pages. If a record exceeds 8192 bytes, the record will be split across two different records.

Because the versioned record is stored as binary, there are no problems with different collations from different databases. Use **sys.dm_tran_version_store_space_usage** to monitor and plan tempdb size based on the version store space usage of databases in a SQL Server instance.

COLUMN NAME	DATA TYPE	DESCRIPTION
database_id	**int**	Database ID of the database.
reserved_page_count	**bigint**	Total count of the pages reserved in tempdb for version store records of the database.
reserved_space_kb	**bigint**	Total space used in kilobytes in tempdb for version store records of the database.

Permissions

On SQL Server, requires `VIEW SERVER STATE` permission.

Examples

The following query can be used to determine space consumed in tempdb, by version store of each database in a SQL Server instance.

```
SELECT
  DB_NAME(database_id) as 'Database Name',
  reserved_page_count,
  reserved_space_kb
FROM sys.dm_tran_version_store_space_usage;
```

Here is the result set.

```
Database Name            reserved_page_count reserved_space_kb
------------------------ -------------------- -----------
msdb                      0                    0
AdventureWorks2016        10                   80
AdventureWorks2016DW      0                    0
WideWorldImporters        20                   160
```

See Also

Dynamic Management Views and Functions (Transact-SQL)
Transaction Related Dynamic Management Views and Functions (Transact-SQL)