cluster() (scope function)

Article • 01/16/2023

Changes the reference of the query to a remote cluster. To access a database within the same cluster, use the database() function. For more information, see cross-database and cross-cluster queries.

Syntax

`cluster(`*name*`)`

Parameters

Name	Type	Required	Description
name	string	✓	The name of the cluster to reference. The value can be specified as a fully qualified domain name, or the name of the cluster without the `.kusto.windows.net` suffix. The value can't be the result of subquery evaluation.

Examples

Use cluster() to access remote cluster

The following query can be run on any cluster.

Run the query

Kusto

```
cluster('help').database('Samples').StormEvents | count

cluster('help.kusto.windows.net').database('Samples').StormEvents | count
```

Output

Count
59066

Use cluster() inside let statements

The previous query can be rewritten to use a query-defined function (`let` statement) that takes a parameter called `clusterName` and passes it to the `cluster()` function.

Run the query

Kusto

```
let foo = (clusterName:string)
{
    cluster(clusterName).database('Samples').StormEvents | count
};
foo('help')
```

Output

Count
59066

Use cluster() inside Functions

The same query as above can be rewritten to be used in a function that receives a parameter `clusterName` - which is passed into the cluster() function.

Kusto

```
.create function foo(clusterName:string)
{
    cluster(clusterName).database('Samples').StormEvents | count
};
```

> ⓘ Note
>
> Stored functions using the `cluster()` function can't be used in cross-cluster queries.

Feedback

database() (scope function)

Article • 03/09/2023

Changes the reference of the query to a specific database within the cluster scope.

> ⓘ Note
>
> - For more information, see **cross-database and cross-cluster queries**.
> - For accessing remote cluster and remote database, see **cluster()** scope function.

Syntax

`database(` *databaseName* `)`

Parameters

Name	Type	Required	Description
databaseName	string	The name of the database to reference. The *databaseName* can be either the `DatabaseName` or `PrettyName`. The argument must be a constant value and can't come from a subquery evaluation.	

Examples

Use database() to access table of other database

Run the query

Kusto

```
database('Samples').StormEvents | count
```

Output

Count

Count
59066

Use database() inside let statements

The query above can be rewritten as a query-defined function (let statement) that receives a parameter `dbName` - which is passed into the database() function.

Kusto

```
let foo = (dbName:string)
{
    database(dbName).StormEvents | count
};
foo('help')
```

Output

Count
59066

Use database() inside stored functions

The same query as above can be rewritten to be used in a function that receives a parameter `dbName` - which is passed into the database() function.

Kusto

```
.create function foo(dbName:string)
{
    database(dbName).StormEvents | count
};
```

> ⓘ Note
>
> Such functions can be used only locally and not in the cross-cluster query.

Feedback

external_table()

Article • 03/12/2023

References an external table by name.

> ⓘ Note
>
> The `external_table` function has similar restrictions as the **table** function. Standard **query limits** apply to external table queries as well.

Syntax

`external_table(` *TableName* [`,` *MappingName*] `)`

Parameters

Name	Type	Required	Description
TableName	string	✓	The name of the external table being queried. Must reference an external table of kind `blob`, `adl`, or `sql`.
MappingName	string		A name of a mapping object that maps fields in the external data shards to columns output.

Authentication and authorization

The authentication method to access an external table is based on the connection string provided during its creation, and the permissions required to access the table vary depending on the authentication method. For more information, see Azure Storage external table or SQL Server external table.

Next steps

- External tables overview
- Create and alter Azure Storage external tables
- Create and alter SQL Server external tables.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

materialize()

Article • 01/26/2023

Captures the value of a tabular expression for the duration of the query execution so that it can be referenced multiple times by the query without recalculation.

Syntax

`materialize(`*expression*`)`

Parameters

Name	Type	Required	Description
expression	string	✓	The tabular expression to be evaluated and cached during query execution.

Remarks

The `materialize()` function is useful in the following scenarios:

- To speed up queries that perform *heavy* calculations whose results are used multiple times in the query.
- To evaluate a tabular expression only once and use it many times in a query. This is commonly required if the tabular expression is non-deterministic. For example, if the expression uses the `rand()` or the `dcount()` functions.

> ⓘ Note
>
> Materialize has a cache size limit of **5 GB**. This limit is per cluster node and is mutual for all queries running concurrently. If a query uses `materialize()` and the cache can't hold any more data, the query will abort with an error.

> 💡 Tip
>
> Another way to perform materialization of tabular expression is by using the `hint.materialized` flag of the **as operator** and **partition operator**. They all share a single materialization cache.

Tip

- Push all possible operators that reduce the materialized data set and keep the semantics of the query. For example, use common filters on top of the same materialized expression.
- Use materialize with join or union when their operands have mutual subqueries that can be executed once. For example, join/union fork legs. See **example of using join operator**.
- Materialize can only be used in let statements if you give the cached result a name. See **example of using let statements**).

Examples of query performance improvement

The following example shows how `materialize()` can be used to improve performance of the query. The expression `_detailed_data` is defined using `materialize()` function and therefore is calculated only once.

Run the query

Kusto

```
let _detailed_data = materialize(StormEvents | summarize Events=count() by
State, EventType);
_detailed_data
| summarize TotalStateEvents=sum(Events) by State
| join (_detailed_data) on State
| extend EventPercentage = Events*100.0 / TotalStateEvents
| project State, EventType, EventPercentage, Events
| top 10 by EventPercentage
```

Output

State	EventType	EventPercentage	Events
HAWAII WATERS	Waterspout	100	2
LAKE ONTARIO	Marine Thunderstorm Wind	100	8
GULF OF ALASKA	Waterspout	100	4
ATLANTIC NORTH	Marine Thunderstorm Wind	95.2127659574468	179

State	EventType	EventPercentage	Events
LAKE ERIE	Marine Thunderstorm Wind	92.5925925925926	25
E PACIFIC	Waterspout	90	9
LAKE MICHIGAN	Marine Thunderstorm Wind	85.1648351648352	155
LAKE HURON	Marine Thunderstorm Wind	79.3650793650794	50
GULF OF MEXICO	Marine Thunderstorm Wind	71.7504332755633	414
HAWAII	High Surf	70.0218818380744	320

The following example generates a set of random numbers and calculates:

- How many distinct values in the set (`Dcount`)
- The top three values in the set
- The sum of all these values in the set

This operation can be done using batches and materialize:

Run the query

Kusto

```
let randomSet =
    materialize(
        range x from 1 to 3000000 step 1
        | project value = rand(10000000));
randomSet | summarize Dcount=dcount(value);
randomSet | top 3 by value;
randomSet | summarize Sum=sum(value)
```

Result set 1:

Dcount
2578351

Result set 2:

value
9999998
9999998

value
9999997

Result set 3:

Sum
15002960543563

Examples of using materialize()

> **Tip**
>
> Materialize your column at ingestion time if most of your queries extract fields from dynamic objects across millions of rows.

To use the `let` statement with a value that you use more than once, use the materialize() function. Try to push all possible operators that will reduce the materialized data set and still keep the semantics of the query. For example, use filters, or project only required columns.

Kusto

```
let materializedData = materialize(Table
| where Timestamp > ago(1d));
union (materializedData
| where Text !has "somestring"
| summarize dcount(Resource1)), (materializedData
| where Text !has "somestring"
| summarize dcount(Resource2))
```

The filter on `Text` is mutual and can be pushed to the materialize expression. The query only needs columns `Timestamp`, `Text`, `Resource1`, and `Resource2`. Project these columns inside the materialized expression.

Kusto

```
let materializedData = materialize(Table
| where Timestamp > ago(1d)
| where Text !has "somestring"
| project Timestamp, Resource1, Resource2, Text);
union (materializedData
```

```
    | summarize dcount(Resource1)), (materializedData
    | summarize dcount(Resource2))
```

If the filters aren't identical, as in the following query:

Kusto

```
let materializedData = materialize(Table
| where Timestamp > ago(1d));
union (materializedData
| where Text has "String1"
| summarize dcount(Resource1)), (materializedData
| where Text has "String2"
| summarize dcount(Resource2))
```

When the combined filter reduces the materialized result drastically, combine both filters on the materialized result by a logical `or` expression as in the following query. However, keep the filters in each union leg to preserve the semantics of the query.

Kusto

```
let materializedData = materialize(Table
| where Timestamp > ago(1d)
| where Text has "String1" or Text has "String2"
| project Timestamp, Resource1, Resource2, Text);
union (materializedData
| where Text has "String1"
| summarize dcount(Resource1)), (materializedData
| where Text has "String2"
| summarize dcount(Resource2))
```

Feedback

materialized_view() function

Article • 03/15/2023

References the materialized part of a materialized view.

The `materialized_view()` function supports a way of querying the *materialized* part only of the view, while specifying the max latency the user is willing to tolerate. This option isn't guaranteed to return the most up-to-date records, but should always be more performant than querying the entire view. This function is useful for scenarios in which you're willing to sacrifice some freshness for performance, for example in telemetry dashboards.

Syntax

`materialized_view(` *ViewName* `,` [*max_age*] `)`

Parameters

Name	Type	Required	Description
ViewName	string	✓	The name of the materialized view.
max_age	int		If not provided, only the *materialized* part of the view is returned. If provided, the function will return the *materialized* part of the view if last materialization time is greater than `@now - max_age`. Otherwise, the entire view is returned, which is identical to querying *ViewName* directly.

Examples

Query the *materialized* part of the view only, independent on when it was last materialized.

Kusto

```
materialized_view("ViewName")
```

Query the *materialized* part only if it was materialized in the last 10 minutes. If the materialized part is older than 10 minutes, return the full view. This option is expected to be less performant than querying the materialized part.

Kusto

```
materialized_view("ViewName", 10m)
```

Notes

- Once a view is created, it can be queried just as any other table in the database, including participate in cross-cluster / cross-database queries.
- Materialized views aren't included in wildcard unions or searches.
- Syntax for querying the view is the view name (like a table reference).
- Querying the materialized view will always return the most up-to-date results, based on all records ingested to the source table. The query combines the materialized part of the view with all unmaterialized records in the source table. For more information, see how materialized views work for details.

Feedback

Query results cache

Article • 01/16/2023

Kusto includes a query results cache. You can choose to get cached results when issuing a query. You'll experience better query performance and lower resource consumption if your query's results can be returned by the cache. However, this performance comes at the expense of some "staleness" in the results.

Use the cache

Set the `query_results_cache_max_age` option as part of the query to use the query results cache. You can set this option in the query text or as a client request property. For example:

Kusto

```
set query_results_cache_max_age = time(5m);
GithubEvent
| where CreatedAt > ago(180d)
| summarize arg_max(CreatedAt, Type) by Id
```

The option value is a `timespan` that indicates the maximum "age" of the results cache, measured from the query start time. Beyond the set timespan, the cache entry is obsolete and won't be used again. Setting a value of 0 is equivalent to not setting the option.

Compatibility between queries

Identical queries

The query results cache returns results only for queries that are considered "identical" to a previous cached query. Two queries are considered identical if all of the following conditions are met:

- The two queries have the same representation (as UTF-8 strings).
- The two queries are made to the same database.
- The two queries share the same client request properties. The following properties are ignored for caching purposes:
  - ClientRequestId
  - Application

- User

Incompatible queries

The query results won't be cached if any of the following conditions is true:

- The query references a table that has the RestrictedViewAccess policy enabled.
- The query references a table that has the RowLevelSecurity policy enabled.
- The query uses any of the following functions:
  - current_principal
  - current_principal_details
  - current_principal_is_member_of
- The query accesses an external table or an external data.
- The query uses the evaluate plugin operator.

No valid cache entry

If a cached result satisfying the time constraints couldn't be found, or there isn't a cached result from an "identical" query in the cache, the query will be executed and its results cached, as long as:

- The query execution completes successfully, and
- The query results size doesn't exceed 16 MB.

Results from the cache

How does the service indicate that the query results are being served from the cache? When responding to a query, Kusto sends another ExtendedProperties response table that includes a `Key` column and a `Value` column. Cached query results will have another row appended to that table:

- The row's `Key` column will contain the string `ServerCache`
- The row's `Value` column will contain a property bag with two fields:
  - `OriginalClientRequestId` - Specifies the original request's ClientRequestId.
  - `OriginalStartedOn` - Specifies the original request's execution start time.

Distribution

The cache isn't shared by cluster nodes. Every node has a dedicated cache in its own private storage. If two identical queries land on different nodes, the query will be

executed and cached on both nodes. This process can happen if weak consistency is used. By setting query consistency to `affinitizedweakconsistency`, you can have weakly consistency queries that are identical land on the same query head, and thus increase the cache hit rate.

Management

The following management and observability commands are supported:

- Show query results cache: Returns statistics related to the query results cache.
- Clear query results cache: Clears query results cache.
- Refresh query cache entry: a specific query cache entry can be refreshed using `query_results_cache_force_refresh` (OptionQueryResultsCacheForceRefresh)client request property. When set to `true`, this command will force query results cache to be refreshed also when an existing cache is present. This process is useful in scenarios that require queries results to be available for querying. This property must be used in combination with 'query_results_cache_max_age', and sent via ClientRequestProperties object. The property can't be part of a 'set' statement.

Capacity

The cache capacity is currently fixed at 1 GB per cluster node. The eviction policy is LRU.

Shard level query results cache

The query results cache is effective when the exact same query is run multiple times in rapid succession and can tolerate returning slightly old data. However, some scenarios, like a live dashboard, require the most up-to-date results.

For example, a query that runs every 10 seconds and spans the last 1 hour can benefit from caching intermediate query results at the storage (shard) level.

> ⓘ **Note**
>
> This feature is only available on EngineV3 clusters.

The shard level query results cache is automatically enabled when the `Query results cache` is in use. Because it shares the same cache as `Query results cache`, the same capacity and eviction policies apply.

Syntax

`set` `query_results_cache_per_shard`; *Query*

> **Note**
>
> This option can be set in the query text or as a **client request property**.

Example

Kusto

```
set query_results_cache_per_shard;
GithubEvent
| where CreatedAt > ago(180d)
| summarize arg_max(CreatedAt, Type) by Id
```

Feedback

table() (scope function)

Article • 05/23/2023

The table() function references a table by providing its name as an expression of type `string`.

Syntax

`table(` *TableName* [`,` *DataScope*] `)`

Parameters

Name	Type	Required	Description
TableName	string	✓	The name of the table being referenced. The value of this expression must be constant at the point of call to the function, meaning it cannot vary by the data context.
DataScope	string		Used to restrict the table reference to data according to how this data falls under the table's effective cache policy. If used, the actual argument must be one of the Valid data scope values.

Valid data scope values

Value	Description
`hotcache`	Only data that is categorized as hot cache will be referenced.
`all`	All the data in the table will be referenced.
`default`	The default is `all`, except if it has been set to `hotcache` by the cluster admin.

Returns

`table(T)` returns:

- Data from table *T* if a table named *T* exists.
- Data returned by function *T* if a table named *T* doesn't exist but a function named *T* exists. Function *T* must take no arguments and must return a tabular result.
- A semantic error is raised if there's no table named *T* and no function named *T*.

Examples

Use table() to access table of the current database

Run the query

Kusto

```
table('StormEvents') | count
```

Output

Count
59066

Use table() inside let statements

The query above can be rewritten as a query-defined function (let statement) that receives a parameter `tableName` - which is passed into the table() function.

Run the query

Kusto

```
let foo = (tableName:string)
{
    table(tableName) | count
};
foo('StormEvents')
```

Output

Count
59066

Use table() inside Functions

The same query as above can be rewritten to be used in a function that receives a parameter `tableName` - which is passed into the table() function.

Kusto

```
.create function foo(tableName:string)
{
    table(tableName) | count
};
```

> **Note**
>
> Such functions can be used only locally and not in the cross-cluster query.

Use table() with non-constant parameter

A parameter, which isn't a scalar constant string, can't be passed as a parameter to the `table()` function.

Below, given an example of workaround for such case.

Run the query

Kusto

```
let T1 = print x=1;
let T2 = print x=2;
let _choose = (_selector:string)
{
    union
    (T1 | where _selector == 'T1'),
    (T2 | where _selector == 'T2')
};
_choose('T2')
```

Output

x
2

Feedback

Was this page helpful? Yes No

toscalar()

Article • 03/16/2023

Returns a scalar constant value of the evaluated expression.

This function is useful for queries that require staged calculations. For example, calculate a total count of events, and then use the result to filter groups that exceed a certain percent of all events.

Any two statements must be separated by a semicolon.

Syntax

`toscalar(`*expression*`)`

Parameters

Name	Type	Required	Description
expression	string	✓	The value to convert to a scalar value.

Returns

A scalar constant value of the evaluated expression. If the result is a tabular, then the first column and first row will be taken for conversion.

> Tip
>
> You can use a **let statement** for readability of the query when using `toscalar()`.

Limitations

`toscalar()` can't be applied on a scenario that applies the function on each row. This is because the function can only be calculated a constant number of times during the query execution. Usually, when this limitation is hit, the following error will be returned: `can't use '<column name>' as it is defined outside its row-context scope.`

In the following example, the query fails with the error:

> `'toscalar': can't use 'x' as it is defined outside its row-context scope.`

Kusto

```
let _dataset1 = datatable(x:long)[1,2,3,4,5];
let _dataset2 = datatable(x:long, y:long) [ 1, 2, 3, 4, 5, 6];
let tg = (x_: long)
{
    toscalar(_dataset2| where x == x_ | project y);
};
_dataset1
| extend y = tg(x)
```

This failure can be mitigated by using the `join` operator, as in the following example:

Run the query

Kusto

```
let _dataset1 = datatable(x: long)[1, 2, 3, 4, 5];
let _dataset2 = datatable(x: long, y: long) [1, 2, 3, 4, 5, 6];
_dataset1
| join (_dataset2) on x
| project x, y
```

Output

x	y
1	2
3	4
5	6

Examples

Evaluate `Start`, `End`, and `Step` as scalar constants, and use the result for `range` evaluation.

Run the query

Kusto

```
let Start = toscalar(print x=1);
let End = toscalar(range x from 1 to 9 step 1 | count);
let Step = toscalar(2);
range z from Start to End step Step | extend start=Start, end=End, step=Step
```

Output

z	start	end	step
1	1	9	2
3	1	9	2
5	1	9	2
7	1	9	2
9	1	9	2

The following example shows how `toscalar` can be used to "fix" an expression so that it will be calculated precisely once. In this case, the expression being calculated returns a different value per evaluation.

Run the query

Kusto

```
let g1 = toscalar(new_guid());
let g2 = new_guid();
range x from 1 to 2 step 1
| extend x=g1, y=g2
```

Output

x	y
e6a15e72-756d-4c93-93d3-fe85c18d19a3	c2937642-0d30-4b98-a157-a6706e217620
e6a15e72-756d-4c93-93d3-fe85c18d19a3	c6a48cb3-9f98-4670-bf5b-589d0e0dcaf5

Feedback