Entity types

Article • 03/07/2022

Kusto queries execute in the context of some Kusto database that is attached to a Kusto cluster. Data in the database is arranged in tables, which the query may reference, and within the table it is organized as a rectangular grid of columns and rows. Additionally, queries may reference stored functions in the database, which are query fragments made available for reuse.

- Clusters are entities that hold databases. Clusters have no name, but they can be referenced by using the `cluster()` special function with the cluster's URI. For example, `cluster("https://help.kusto.windows.net")` is a reference to a cluster that holds the `Samples` database.

- Databases are named entities that hold tables and stored functions. All Kusto queries run in the context of some database, and the entities of that database may be referenced by the query with no qualifications. Additionally, other databases of the cluster, or databases or other clusters, may be referenced using the database() special function. For example, `cluster("https://help.kusto.windows.net").database("Samples")` is a universal reference to a specific database.

- Tables are named entities that hold data. A table has an ordered set of columns, and zero or more rows of data, each row holding one data value for each of the columns of the table. Tables may be referenced by name only if they are in the database in context of the query, or by qualifying them with a database reference otherwise. For example, `cluster("https://help.kusto.windows.net").database("Samples").StormEvents` is a universal reference to a particular table in the `Samples` database. Tables may also be referenced by using the table() special function.

- Columns are named entities that have a scalar data type. Columns are referenced in the query relative to the tabular data stream that is in context of the specific operator referencing them.

- Stored functions are named entities that allow reuse of Kusto queries or query parts.

- Views are virtual tables based on functions (stored or defined in an ad-hoc fashion).

- External tables are entities that reference data stored outside Kusto database. External tables are used for exporting data from Kusto to external storage as well as for querying external data without ingesting it into Kusto.

Entity names

Article • 01/24/2023

Kusto entities (databases, tables, columns, and stored functions; clusters are an exception) are named. The name of an entity identifies the entity, and is guaranteed to be unique in the scope of its container given its type. (So, for example, two tables in the same database can't have the same name, but a table and a database may have the same name because they aren't in the same scope, and a table and a stored function may have the same name because they aren't of the same entity type.)

Entity names are **case-sensitive** for resolving purposes (so, for example, you can't refer to a table called `ThisTable` as `thisTABLE`).

Entity names are one example of **identifiers**. Other identifiers include the names of parameters to functions and binding a name through a let statement.

Entity pretty names

Some entities (such as databases) may have, in addition to their entity name, a **pretty name**. Pretty names can be used to reference the entity in queries (like entity names), but, unlike entity names, pretty names aren't necessarily unique in the context of their container. When a container has multiple entities with the same pretty name, the pretty name can't be used to reference the entity.

Pretty names allow middle-tier applications to map automatically created entity names (such as UUIDs) to names that are human-readable for display and referencing purposes.

Identifier naming rules

Identifiers are used to name various entities (entities or otherwise). Valid identifier names follow these rules:

- They have to be between 1 and 1024 characters long.
- They may contain letters, digits, underscores (`_`), spaces, dots (`.`), and dashes (`-`).
  - Identifiers consisting only of letters, digits, and underscores don't require quoting when the identifier is being referenced.
  - Identifiers containing at least one of (spaces, dots, or dashes) do require quoting (see below).
- They're case-sensitive.

ⓘ Note

The `$` character isn't allowed in stored entity names unless generated by a KQL operator.

Identifier quoting

Identifiers that are identical to some query language keywords, or have one of the special characters noted above, require quoting when they're referenced directly by a query:

Query text	Comments
`entity`	Entity names (`entity`) that don't include special characters or map to some language keyword require no quoting
`['entity-name']`	Entity names that include special characters (here: `-`) must be quoted using `['` and `']` or using `["` and `"]`
`["where"]`	Entity names that are language keywords must be quoted using `['` and `']` or using `["` and `"]`

Naming your entities to avoid collisions with Kusto language keywords

As the Kusto Query Language includes a number of keywords that have the same naming rules as identifiers, it's possible to have entity names that are actually keywords, but then referring to these names becomes difficult (one must quote them).

Alternatively, one might want to choose entity names that are guaranteed to never "collide" with a Kusto keyword. The following guarantees are made:

1. The Kusto Query Language won't define a keyword that starts with a capital letter (`A` to `Z`).
2. The Kusto Query Language won't define a keyword that starts with a single underscore (`_`).
3. The Kusto Query Language won't define a keyword that ends with a single underscore (`_`).

The Kusto Query Language reserves all identifiers that start or end with a sequence of two underscore characters (`__`); users can't define such names for their own use.

Entity references

Article • 03/07/2022

Reference Kusto schema entities in a query by using their names. Valid entity names include *databases*, *tables*, *columns*, and stored functions. *Clusters* can't be referenced by their names. If the entity's container is unambiguous in the current context, use the entity name without additional qualifications. For example, when running a query against a database called `DB`, you may reference a table called `T` in that database by its name, `T`.

If the entity's container isn't available from the context, or you want to reference an entity from a container different than the container in context, use the entity's **qualified name**. The name is the concatenation of the entity name to the container's, and potentially its container's, and so on. In this way, a query running against database `DB` may refer to a table `T1` in a different database `DB1` of the same cluster, by using `database("DB1").T1`. If the query wants to reference a table from another cluster it can do so, for example, by using `cluster("https://C2.kusto.windows.net/").database("DB2").T2`.

Entity references can also use the entity pretty name, as long as it's unique in the context of the entity's container. For more information, see entity pretty names.

Wildcard matching for entity names

In some contexts, you may use a wildcard (`*`) to match all or part of an entity name. For example, the following query references all tables in the current database, and all tables in database `DB` whose name starts with a `T`:

Kusto

```
union *, database("DB1").T*
```

ⓘ **Note**

Wildcard matching can't match entity names that start with a dollar sign (`$`). Such names are system-reserved.

Next steps

- schema entity types
- schema entity names

Databases

Article • 03/07/2022

Kusto follows a relation model of storing the data where upper-level entity is a `database`.

Kusto cluster can host several databases, where each database will host its own collection of tables, stored functions, and external tables. Each database has its own permissions set, based on Role Based Access Control (RBAC) model

Notes

- Database names must follow the rules for entity names.
- Maximum limit of databases per cluster is 10,000.
- Queries combining data from multiple tables in the same database and queries combining data from multiple databases in the same cluster have comparable performance.

Tables

Article • 05/30/2023

Tables are named entities that hold data. A table has an ordered set of columns, and zero or more rows of data. Each row holds one data value for each of the columns of the table. The order of rows in the table is unknown, and doesn't in general affect queries, except for some tabular operators (such as the top operator) that are inherently undetermined.

Tables occupy the same namespace as stored functions. If a stored function and a table both have the same name, the stored function will be chosen.

Notes

- Table names are case-sensitive.
- Table names follow the rules for entity names.
- Maximum limit of tables per database is 10,000.

Details on how to create and manage tables can be found under managing tables

Table References

The simplest way to reference a table is by using its name. This reference can be done for all tables that are in the database in context. For example, the following query counts the records of the current database's `StormEvents` table:

Kusto

```
StormEvents
| count
```

An equivalent way to write the query above is by escaping the table name:

Kusto

```
["StormEvents"]
| count
```

Tables may also be referenced by explicitly noting the database (or database and cluster) they are in. Then you can author queries that combine data from multiple databases and clusters. For example, the following query will work with any database in context, as long as the caller has access to the target database:

Kusto

```
cluster("https://help.kusto.windows.net").database("Samples").StormEvents
| count
```

It's also possible to reference a table by using the table() special function, as long as the argument to that function evaluates to a constant. For example:

Kusto

```
let counter=(TableName:string) { table(TableName) | count };
counter("StormEvents")
```

Note

Use the `table()` special function to explicitly specify the table data scope. For example, use this function to restrict processing to the data in the table that falls in the hot cache.

See also

- Estimate table size

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Columns

Article • 03/07/2022

Every table in Kusto, and every tabular data stream, is a rectangular grid of columns and rows. Every column in the table has a name and a specific scalar data type. The columns of a table or a tabular data stream are ordered, so a column also has a specific position in the table's collection of columns.

Notes

- Column names are case-sensitive.
- Column names follow the rules for entity names.
- Maximum limit of columns per table is 10,000.

In queries, columns are generally referenced by name only. They can only appear in expressions, and the query operator under which the expression appears determines the table or tabular data stream, so the column's name need not be further scoped. For example, in the following query we have an unnamed tabular data stream (defined through the datatable operator that has a single column, `c`. The tabular data stream is then filtered by a predicate on the value of that column, producing a new unnamed tabular data stream with the same columns but fewer rows. The as operator then names the tabular data stream and its value is returned as the results of the query. Note in particular how the column `c` is referenced by name without a need to reference its container (indeed, that container has no name):

Kusto

```
datatable (c:int) [int(-1), 0, 1, 2, 3]
| where c*c >= 2
| as Result
```

> As is often common in the relational databases world, rows are sometimes called **records** and columns are sometimes called **attributes**.

Details on managing columns can be found under managing columns.

Feedback

Was this page helpful? Yes No

Stored functions

Article • 06/05/2023

Functions are reusable queries or query parts. Functions can be stored as database entities, similar to tables, called *stored functions*. Alternatively, functions can be created in an ad-hoc fashion with a let statement, called *query-defined functions*. For more information, see user-defined functions.

To create and manage stored functions, see the Stored functions management overview.

Note

Designate a stored function as a **stored view** to enable the function to participate in `search` and `union *` scenarios.

Views

Article • 06/05/2023

Views are virtual tables based on the result-set of a Kusto Query Language query. Just like a real table, a view contains rows and columns. Unlike a real table, a view doesn't hold its own data storage.

Define a view

Views are defined through user-defined functions, either stored or query-defined, with the following requirements:

- The result of the function must be tabular, meaning it can't be a scalar value.
- The function must take no arguments.

Note

Views are not technically schema entities. However, all functions that comply with the constraints above are regarded as views.

Stored views

When a stored function is designated as a view, it's considered a stored view. Stored views behave like stored functions and can participate in `search` and `union *` scenarios.

To designate a stored function as a stored view, set the `view` property to `true` when you create the function. For more information, see .create function.

Example: Define and use a view

The following query defines and uses a view. The view is used as-if a table called `T` was defined (there's no need to reference the function `T` using the function call syntax `T()`):

Run the query

Kusto

```
let T=() {print x=1, y=2};
T
```

Returns

x	y
1	2

The view keyword

By default, operators that support a wildcard syntax to specify table names will not reference views, even if the view's name matches the wildcard. An example of this type of operator is the union operator. In this case, use the `view` keyword to have the view included as well.

Example: Use the view keyword

For example, the results of the following query include the `T1` view, but not `T2`:

Run the query

Kusto

```
let T1=view (){print Name="T1"};
let T2=(){print Name="T2"};
union T*
```

External tables

Article • 05/23/2023

An **external table** is a schema entity that references data stored outside a database in your cluster.

Similar to tables, an external table has a well-defined schema (an ordered list of column name and data type pairs). Unlike tables where data is ingested into your cluster, external tables operate on data stored and managed outside your cluster.

Supported external data stores are:

- Files stored in Azure Blob Storage or in Azure Data Lake. Most commonly the data is stored in some standard format such as CSV, JSON, Parquet, AVRO, etc. For the list of supported formats, refer to supported formats.
- SQL Server table.

See the following ways of creating external tables:

- Create or alter Azure Blob Storage/ADLS external tables
- Create or alter delta external tables
- Create and alter SQL Server external tables
- Create external table using Azure Data Explorer web UI Wizard

An **external table** can be referenced by its name using the external_table() function.

Use the following commands to manage external tables:

- .drop external table
- .show external tables
- .show external table schema

For more information about how to query external tables, and ingested and uningested data, see Query data in Azure Data Lake using Azure Data Explorer.

Notes

- External table names:
  - Case-sensitive.
  - Can't overlap with Kusto table names.
  - Follow the rules for entity names.
- Maximum limit of external tables per database is 1,000.
- Azure Data Explorer supports export and continuous export to an external table.

- Data purge isn't applied on external tables. Records are never deleted from external tables.

Fact and dimension tables

Article • 03/07/2022

When designing the schema for an Azure Data Explorer database, think of tables as broadly belonging to one of two categories.

- Fact tables
- Dimension tables

Fact tables

Fact tables are tables whose records are immutable "facts", such as service logs and measurement information. Records are progressively appended into the table in a streaming fashion or in large chunks. The records stay there until they're removed because of cost or because they've lost their value. Records are otherwise never updated.

Entity data is sometimes held in fact tables, where the entity data changes slowly. For example, data about some physical entity, such as a piece of office equipment that infrequently changes location. Since data in Kusto is immutable, the common practice is to have each table hold two columns:

- An identity (`string`) column that identifies the entity
- A last-modified (`datetime`) timestamp column

Only the last record for each entity identity is then retrieved.

Dimension tables

Dimension tables:

- Hold reference data, such as lookup tables from an entity identifier to its properties
- Hold snapshot-like data in tables whose entire contents change in a single transaction

Dimension tables aren't regularly ingested with new data. Instead, the entire data content is updated at once, using operations such as .set-or-replace, .move extents, or .rename tables.

Sometimes, dimension tables might be derived from fact tables. This process can be done via an update policy on the fact table, with a query on the table that takes the last record for each entity.

Differentiate fact and dimension tables

There are processes in Kusto that differentiate between fact tables and dimension tables. One of them is continuous export.

These mechanisms are guaranteed to process data in fact tables precisely once. They rely on the database cursor mechanism.

For example, every execution of a continuous export job, exports all records that were ingested since the last update of the database cursor. Continuous export jobs must differentiate between fact tables and dimension tables. Fact tables only process newly ingested data, and dimension tables are used as lookups. As such, the entire table must be taken into account.

There's no way to "mark" a table as being a "fact table" or a "dimension table". The way data is ingested into the table, and how the table is used, is what identifies its type.