Query best practices

Article • 09/08/2022 • 2 minutes to read

Here are several best practices to follow to make your query run faster.

Action	Use	Don't use	Notes
Time filters	Use time filters first.		Kusto is highly optimized to use time filters.
String operators	Use the `has` operator	Don't use `contains`	When looking for full tokens, `has` works better, since it doesn't look for substrings.
Case-sensitive operators	Use `==`	Don't use `=~`	Use case-sensitive operators when possible.
	Use `in`	Don't use `in~`	
	Use `contains_cs`	Don't use `contains`	If you can use `has` / `has_cs` and not use `contains` / `contains_cs`, that's even better.
Searching text	Look in a specific column	Don't use `*`	`*` does a full text search across all columns.
Extract fields from dynamic objects across millions of rows	Materialize your column at ingestion time if most of your queries extract fields from dynamic objects across millions of rows.		This way, you'll only pay once for column extraction.
Lookup for rare keys/values in dynamic objects	Use `MyTable | where DynamicColumn has "Rare value" | where DynamicColumn.SomeKey == "Rare value"`	Don't use `MyTable | where DynamicColumn.SomeKey == "Rare value"`	This way, you filter out most records, and do JSON parsing only of the rest.
`let` statement with a value that you use	Use the materialize() function		For more information on how to use `materialize()`, see materialize(). For more information, see Optimize

Action	Use	Don't use	Notes
more than once			queries that use named expressions.
Apply conversions on more than 1 billion records	Reshape your query to reduce the amount of data fed into the conversion.	Don't convert large amounts of data if it can be avoided.	
New queries	Use `limit [small number]` or `count` at the end.		Running unbound queries over unknown data sets may yield GBs of results to be returned to the client, resulting in a slow response and a busy cluster.
Case-insensitive comparisons	Use `Col =~ "lowercasestring"`	Don't use `tolower(Col) == "lowercasestring"`	
Compare data already in lowercase (or uppercase)	`Col == "lowercasestring"` (or `Col == "UPPERCASESTRING"`)	Avoid using case insensitive comparisons.	
Filtering on columns	Filter on a table column.	Don't filter on a calculated column.	
	Use `T | where predicate(<expression>)`	Don't use `T | extend _value = <expression> | where predicate(_value)`	
summarize operator	Use the hint.shufflekey=<key> when the `group by keys` of the summarize operator are with high cardinality.		High cardinality is ideally above 1 million.
join operator	Select the table with the fewer rows to be the first one (left-most in query).		
	Use `in` instead of left semi `join` for filtering by a		

Action	Use	Don't use	Notes
	single column.		
Join across clusters	Across clusters, run the query on the "right" side of the join, where most of the data is located.		
Join when left side is small and right side is large	Use hint.strategy=broadcast		Small refers to up to 100,000 records.
Join when both sides are too large	Use hint.shufflekey=<key>		Use when the join key has high cardinality.
Extract values on column with strings sharing the same format or pattern	Use the parse operator	Don't use several `extract()` statements.	For example, values like `"Time = <time>, ResourceId = <resourceId>, Duration = <duration>, ...."`
extract() function	Use when parsed strings don't all follow the same format or pattern.		Extract the required values by using a REGEX.
materialize() function	Push all possible operators that will reduce the materialized data set and still keep the semantics of the query.		For example, filters, or project only required columns. For more information, see Optimize queries that use named expressions.
Use materialized views	Use materialized views for storing commonly used aggregations. Prefer using the `materialized_view()` function to query materialized part only		`materialized_view('MV')`

Feedback