Using the Search Optimization Service

The search optimization service can significantly improve the performance of certain types of lookup and analytical queries that use an extensive set of predicates for filtering.

Enterprise Edition Feature

This feature requires Enterprise Edition (or higher). To inquire about upgrading, please contact Snowflake Support.

In this Topic:

Understanding the Search Optimization Service

The search optimization service aims to significantly improve the performance of certain types of queries on tables, including:

- Selective point lookup queries on tables. A point lookup query returns only one or a small number of distinct rows. Use case examples include:
  - Business users who need fast response times for critical dashboards with highly selective filters.
  - Data scientists who are exploring large data volumes and looking for specific subsets of data.
  - Data applications retrieving a small set of results based on an extensive set of filtering predicates.
- Substring and regular expression searches (e.g. LIKE, ILIKE, RLIKE, etc.).
- Queries on fields in VARIANT, OBJECT, and ARRAY columns that use certain types of predicates (equality predicates, IN predicates, predicates that use ARRAY_CONTAINS and ARRAYS_OVERLAP, and predicates that check for NULL values)..
- Queries that use selected geospatial functions with GEOGRAPHY values.

Preview Feature — Open

The following features are in preview and are available to all accounts that are Enterprise Edition (or higher):

- Column configuration.
- Support for substring and regular expression searches.
- Support for fields in VARIANT, OBJECT, and ARRAY columns.
- Support for geospatial functions with GEOGRAPHY objects.

This feature is being rolled out across different regions. For a list of the regions in which this feature is enabled, see Feature Rollout: Search Optimization Service Support for Column Configuration, String Patterns, VARIANT, and GEOGRAPHY.

Once you identify the queries that can benefit from the search optimization service, you can configure search optimization for the columns and tables used in those queries.

How Does the Search Optimization Service Work?

To improve performance for point lookups, the search optimization service relies on a persistent data structure that serves as an optimized search access path.

A maintenance service that runs in the background is responsible for creating and maintaining the search access path:

- When you configure search optimization for a table, the maintenance service creates and populates the search access path with the data needed to perform the lookups.

  The process of populating data can take time, depending on the size of the table. The service does this work in the background and does not block any concurrent operations on the table.

- When data in the table is updated (for example, by loading new data sets or through DML operations), the maintenance service automatically updates the search access path to reflect the changes to the data.

  If queries are run when the search access path hasn't been updated yet, the queries might run slower but will always return up-to-date results.

This search access path and the maintenance service are transparent to the user. You don't need to create a warehouse for the service that maintains the search access path.

However, note that there is a cost for the storage and compute resources for this service. For more details, see Managing the Costs of the Search Optimization Service (in this topic).

Considering Other Solutions for Optimizing Query Performance

The search optimization service is one of several ways to optimize query performance. Related techniques include:

- Clustering a table.
- Creating one or more materialized views (clustered or unclustered).

Each of these has different advantages:

- Clustering a table can improve the performance of any of the following, as long as they are on the clustering key:
  - Range searches.
  - Equality searches.

  However, a table can be clustered on only a single key (which can contain one or more columns or expressions).

- The search optimization service can improve the performance of equality searches as well as other types of searches for supported data types.
- A materialized view can improve the performance of both equality searches and range searches, as well as some sort operations, but only for the subset of rows and columns included in the materialized view. Materialized views can be also

	Storage Cost	Compute Cost
Search Optimization Service	✔	✔
Materialized View	✔	✔
Clustering the Table		✔

What Access Control Privileges Are Needed For the Search Optimization Service?

To add, configure, or remove search optimization for a table, you must have the following privileges:

- You must have OWNERSHIP privilege on the table.
- You must have ADD SEARCH OPTIMIZATION privilege on the schema that contains the table.

```
GRANT ADD SEARCH OPTIMIZATION ON SCHEMA <schema_name> TO ROLE <role>;
```

To use the search optimization service for a query, you just need SELECT privileges on the table.

You do not need any additional privileges. Because search optimization is a table property, it is automatically detected and used (if appropriate) when querying a table.

Identifying the Tables That Benefit From Search Optimization

Search optimization works best to improve the performance of a query when the table is frequently queried on columns other than the primary cluster key.

Determining the Data Types Supported By the Search Optimization Service

The search optimization service currently supports specific types of queries for the following data types:

- Fixed-point numbers (e.g. INTEGER, NUMERIC).
- DATE, TIME, and TIMESTAMP.
- VARCHAR.
- BINARY.
- VARIANT, OBJECT, and ARRAY. (This is a preview feature.)
- GEOGRAPHY. (This is a preview feature.)

Currently, the search optimization service does not support floating point data types or other data types not listed above. Snowflake might add support for more data types in the future.

The search optimization service also does not support collations.

Identifying Queries That Benefit From Search Optimization

Search optimization works best to improve the performance of the following types of queries:

- A query that typically runs for a few seconds or longer.
- A query in which at least one of the columns accessed through the query filter operation has at least 100,000 to 200,000 distinct values.

  To determine the number of distinct values, you can use either of the following:

  - Use `APPROX_COUNT_DISTINCT` to get the approximate number of distinct values:

    ```
    select approx_count_distinct(column1) from table1;
    ```

  - Use `COUNT(DISTINCT <col_name>)` to get the actual number of distinct values:

    ```
    select count(distinct c1), count (distinct c2)  from test_table;
    ```

Search optimization can improve the performance of these types of queries

- Equality or IN Predicates
- Substrings and Regular Expressions
- Fields in VARIANT Columns
- Geospatial Functions
- Conjunctions of Supported Predicates (AND)
- Disjunctions of Supported Predicates (OR)

Equality or IN Predicates

The search optimization service can improve the performance of queries that use:

- Equality predicates (for example, `<column_name> = <constant>`).
- Predicates that use IN (see example).

Substrings and Regular Expressions

The search optimization service can improve the performance of queries with predicates that search for substrings or use regular expressions. This includes predicates that use:

- LIKE
- LIKE ANY
- LIKE ALL
- ILIKE
- ILIKE ANY
- CONTAINS
- ENDSWITH
- STARTSWITH
- SPLIT_PART (in equality predicates)
- RLIKE
- REGEXP
- REGEXP_LIKE

Preview Feature — Open

Support for substring and regular expression searches is a preview feature that is available to all accounts that are Enterprise Edition (or higher).

This feature is being rolled out across different regions. For a list of the regions in which this feature is enabled, see Feature Rollout: Search Optimization Service Support for Column Configuration, String Patterns, VARIANT, and GEOGRAPHY.

The search optimization service can improve performance when searching for substrings that are 5 or more characters long. (More selective substrings can result in better performance.)

For example, the search optimization service ***does not*** use search access paths for the following predicate because the substring is shorter than 5 characters:

```
like '%TEST%'
```

For the following predicate, the search optimization service can optimize this query, using search access paths to search for the substrings for `SEARCH` and `OPTIMIZED`.

```
like '%SEARCH%IS%OPTIMIZED%'
```

In this example, the search optimization service does not use search access paths for `IS` because the substring is shorter than 5 characters.

For queries that use RLIKE, REGEXP, and REGEXP_LIKE:

- The *`<subject>`* argument must be a TEXT column in a table that has search optimization enabled.
- The *`<pattern>`* argument must be a string constant.

For regular expressions, the search optimization service works best when:

- The pattern contains at least one substring literal that is 5 or more characters long.
- The pattern specifies that the substring should appear at least once.

For example, the following pattern specifies that `string` should appear one or more times in the subject:

```
rlike '(string)+'
```

The search optimization service can improve the performance of queries with the following patterns because each predicate specifies that a substring of 5 or more characters must appear at least once. (Note that the first example uses a dollar-quoted string constant to avoid escaping the backslash characters.)

```
rlike '.*country=(Germany|France|Spain).*'
```

```
rlike '.*phone=[0-9]{3}-?[0-9]{3}-?[0-9]{4}.*'
```

In contrast, the search optimization service does not use search access paths for queries with the following patterns:

- Patterns without any substrings:

  ```
  rlike '.*[0-9]{3}-?[0-9]{3}-?[0-9]{4}.*'
  ```

- Patterns that only contain substrings shorter than 5 characters:

  ```
  rlike '.*tel=[0-9]{3}-?[0-9]{3}-?[0-9]{4}.*'
  ```

- Patterns that use the alternation operator where one option is a substring shorter than 5 characters:

  ```
  rlike '.*(option1|option2|opt3).*'
  ```

- Patterns in which the substring is optional:

  ```
  rlike '.*[a-zA-z]+(string)?[0-9]+.*'
  ```

Even when the substring literals are shorter than 5 characters, the search optimization service can still improve query performance if expanding the regular expression produces a substring literal that is 5 characters or longer.

For example, consider the pattern:

```
.*st=(CA|AZ|NV).*(-->){2,4}.*
```

In this example:

- Although the substring literals (e.g. `st=`, `CA`, etc) are shorter than 5 characters, the search optimization service recognizes that the substring `st=CA`, `st=AZ`, or `st=NV` (each of which is 5 characters long) must appear in the text.
- Similarly, even though the substring literal `-->` is shorter than 5 characters, the search optimization service determines that the substring `-->-->` (which is longer than 5 characters) must appear in the text.

The search optimization service can use search access paths to match these substrings, which can improve the performance of the query.

Fields in VARIANT Columns

The search optimization service can improve the performance of point lookup queries on semi-structured data in Snowflake tables (data in VARIANT, OBJECT, and ARRAY columns).

When VARIANT support for the search optimization service is configured for columns in a table, the search optimization service automatically includes VARIANT, OBJECT, and ARRAY columns in a search access path. This even applies to columns where the structure is deeply nested and the structure changes frequently.

The next sections provide more details about this support:

- Supported Data Types for Constants and Casts in Predicates for VARIANT Types
- Support for VARIANT Values Cast as TEXT
- Supported Predicates for VARIANT Types
- Current Limitations in Support for VARIANT Types

Preview Feature — Open

Support for fields in VARIANT, OBJECT, and ARRAY columns is a preview feature that is available to all accounts that are Enterprise Edition (or higher).

This feature is being rolled out across different regions. For a list of the regions in which this feature is enabled, see Feature Rollout: Search Optimization Service Support for Column Configuration, String Patterns, VARIANT, and GEOGRAPHY.

Supported Data Types for Constants and Casts in Predicates for VARIANT Types

The search optimization service can improve the performance of queries of semi-structured data where the following types are used for the constant and the implicit or explicit cast for the element:

- FIXED (including casts that specify a valid precision and scale)

- TIMESTAMP, TIMESTAMP_LTZ, TIMESTAMP_NTZ, TIMESTAMP_TZ (including casts that specify a scale)

The search optimization service supports the casting of types using:

- CAST and the :: operator
- TRY_CAST

Support for VARIANT Values Cast as TEXT

The search optimization service can also improve the performance of queries in which VARIANT columns are cast to TEXT and are compared to constants that are cast to TEXT.

For example, suppose that `src` is a VARIANT column containing boolean, date, and time values that have been converted to VARIANT:

```
create or replace table test_table
(
    id integer,
    src variant
);

insert into test_table select 1, to_variant('true'::boolean);          -- BOOLEAN
insert into test_table select 2, to_variant('2020-01-09'::date);       -- DATE
insert into test_table select 3, to_variant('01:02:03.899213'::time);  -- TIME
```

For this table, the search optimization service can improve the following queries, which cast the VARIANT column to TEXT and compare the column to string constants:

```
select * from test_table where src::text = 'true';
select * from test_table where src::text = '2020-01-09';
select * from test_table where src::text = '01:02:03.899213';
```

Supported Predicates for VARIANT Types

The search optimization service can improve queries with the types of predicates listed below. In the examples below, `src` is the VARIANT column, and `<path_to_variant_field>` is a path to a field in the VARIANT column.

- Equality predicates of the following form:

  `where <path_to_variant_field>[::<target_data_type>] = <constant>`

  `<target_data_type>` (if specified) and the data type of `<constant>` must be one of the supported types listed above.

  Note that `::` is just an example of one of the supported ways of casting the value to a specific type.

  For example, the search optimization service supports:

  - Matching an element against a NUMBER constant without explicitly casting the element.

    ```
    where src:person.age = 42;
    ```

  - Explicitly casting an element to NUMBER with a specified precision and scale.

    ```
    where src:location.temperature::number(8, 6) = 23.456789;
    ```

  - Matching an element against a TEXT constant without explicitly casting the element.

    ```
    where src:sender_info.ip_address = '123.123.123.123';
    ```

  - Explicitly casting an element to TEXT.

    ```
    where src:salesperson.name::text = 'John Appleseed';
    ```

  - Explicitly casting an element to DATE.

    ```
    where src:events.date::date = '2021-03-26';
    ```

```
where src:events.time_info::time(6) = '01:02:03.456789';
```

- Explicitly casting an element to TIMESTAMP with a specified scale.

```
where src:event_logs.exceptions.timestamp_info(3) = '2021-03-26 15:00:00.123 -0800';
```

- Predicates that use the ARRAY functions, such as:
  - `where ARRAY_CONTAINS(<constant>::VARIANT, <path_to_variant_field>)`

    `<constant>` must not be NULL, and the data type of `<constant>` must be one of the supported types listed above.

    Note that `::` is just an example of one of the supported ways of casting the value to a specific type.

    For example:

```
where array_contains('77.146.211.88'::variant, src:logs.ip_addresses)
```

  - `where ARRAYS_OVERLAP(ARRAY_CONSTRUCT(<constant_1>, <constant_2>, ..., <constant_N>), <path_to_variant_field>)`

    The data type of each constant (`<constant_1>`, `<constant_2>`, etc.) must be one of the supported types listed above. The constructed ARRAY can include NULL constants.

    For example:

```
where arrays_overlap(
    array_construct('122.63.45.75', '89.206.83.107'), src:senders.ip_addresses)
```

- The following predicates that check for NULL values:
  - `where IS_NULL_VALUE(<path_to_variant_field>)`

    Note that IS_NULL_VALUE applies to JSON null values and not to SQL NULL values.

  - `where <path_to_variant_field> IS NOT NULL`
  - `where <variant_column> IS NULL`

    where `<variant_column>` refers to the column and not a path to an element in the semi-structured data.

    For example, the search optimization service supports using the VARIANT column `src` but not the path to the field `src:person:age` in that VARIANT column.

Current Limitations in Support for VARIANT Types

Currently, support for VARIANT types in the search optimization service has the following limitations:

- Predicates that use XMLGET are not supported.
- Predicates of the form `<variant_field> IS NULL` are not supported.
- Predicates where the constants are results of scalar subqueries are not supported.
- Predicates that specify paths to elements that contain sub-elements are not supported.

The current limitations of the search optimization service also apply to this feature.

Geospatial Functions

Preview Feature — Open

Support for geospatial functions with GEOGRAPHY objects is a preview feature that is available to all accounts that are Enterprise Edition (or higher).

This feature is being rolled out across different regions. For a list of the regions in which this feature is enabled, see Feature Rollout: Search Optimization Service Support for Column Configuration, String Patterns, VARIANT, and GEOGRAPHY.

The search optimization service can improve the performance of queries with predicates that use geospatial functions with GEOGRAPHY objects. The following sections provide more details:

- Supported Predicates With Geospatial Functions
- Examples That Use Geospatial Functions

Note

Supported Predicates With Geospatial Functions

For queries with predicates that use the following functions:

- ST_INTERSECTS
- ST_CONTAINS
- ST_WITHIN
- ST_DWITHIN
- ST_COVERS
- ST_COVEREDBY

The search optimization service can improve performance if:

- One input expression is a GEOGRAPHY column in a table, and
- The other input expression is a GEOGRAPHY constant (created through a conversion or constructor function).
- For ST_DWITHIN, the distance argument is a non-negative REAL constant.

Because the search optimization service is designed for predicates that are highly selective and because predicates filter by proximity between geospatial objects, clustering geospatial objects by proximity in the table can result in better performance. For example, you can cluster the GEOGRAPHY values by their ST_GEOHASH values or by a combination of the coordinates of the centroid or bounding box of the objects.

Note that this feature has the same limitations that apply to the search optimization service.

Examples That Use Geospatial Functions

The following statements create and configure the table used in the examples in this section:

```
create or replace table geospatial_table (id number, g1 geography);
insert into geospatial_table values
  (1, 'POINT(-122.35 37.55)'),
  (2, 'LINESTRING(-124.20 42.00, -120.01 41.99)'),
  (3, 'POLYGON((0 0, 2 0, 2 2, 0 2, 0 0))');
alter table geospatial_table add search optimization;
```

Examples of Supported Predicates

The following query is an example of a query supported by the search optimization service. The search optimization service can use search access paths to improve the performance of this query:

```
select id from geospatial_table where
  st_intersects(
    g1,
    to_geography('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'));
```

The following are examples of additional predicates that are supported by the search optimization service:

```
...
  st_intersects(
    to_geography('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
    g1)
```

```
...
  st_contains(
    to_geography('POLYGON((-74.17 40.64, -74.1796875 40.58, -74.09 40.58, -74.09 40.64, -74.17 40.64))'),
    g1)
```

```
...
  st_contains(
    g1,
    to_geography('MULTIPOINT((0 0), (1 1))'))
  g1)
```

```
...
  st_within(
    g1,
    to_geography('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'))
```

```
...
  st_covers(
    to_geography('POLYGON((-1 -1, -1 4, 4 4, 4 -1, -1 -1))'),
    g1)
```

```
...
  st_covers(
    g1,
    to_geography('POINT(0 0)'))
```

```
...
  st_coveredby(
    to_geography('POLYGON((1 1, 2 1, 2 2, 1 2, 1 1))'),
    g1)
```

```
...
  st_coveredby(
    g1,
    to_geography('POINT(-122.35 37.55)'))
```

```
...
  st_dwithin(
    to_geography('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
    g1,
    100000)
```

```
...
  st_dwithin(
    g1,
    to_geography('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
    100000)
```

Examples of Constructing GEOGRAPHY Constants

The following are examples of predicates that use different conversion and constructor functions for the GEOGRAPHY constant.

```
...
  st_intersects(
    g1,
    st_geographyfromwkt('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'))
```

```
...
  st_intersects(
    st_geogfromtext('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
    g1)
```

```
...
  st_contains(
    st_geographyfromewkt('POLYGON((-74.17 40.64, -74.1796875 40.58, -74.09 40.58, -74.09 40.64, -74.17 40.64))'),
    g1)
    g1)
```

```
...
  st_covers(
    g1,
    st_makepoint(0.2, 0.8))
```

```
...
  st_intersects(
    g1,
    st_makeline(
      to_geography('MULTIPOINT((0 0), (1 1))'),
      to_geography('POINT(0.8 0.2)')))
```

```
...
  st_intersects(
    st_polygon(
      to_geography('SRID=4326;LINESTRING(0.0 0.0, 1.0 0.0, 1.0 2.0, 0.0 2.0, 0.0 0.0)')),
    g1)
```

```
...
  st_within(
    g1,
    try_to_geography('POLYGON((-1 -1, -1 4, 4 4, 4 -1, -1 -1))'))
```

```
...
  st_covers(
    g1,
    st_geogpointfromgeohash('s00'))
```

Conjunctions of Supported Predicates (AND)

For queries that use conjunctions of predicates (i.e., AND), query performance can be improved by search optimization if any of the predicates adhere to the conditions above.

For example, suppose that a query has:

`where <condition_x> and <condition_y>`

Search optimization can improve performance if either condition separately returns a few rows (i.e., `<condition_x>` returns a few rows ***or*** `<condition_y>` returns a few rows).

If `<condition_x>` returns a few rows but `<condition_y>` returns many rows, the query performance can benefit from search optimization.

See these additional examples.

Disjunctions of Supported Predicates (OR)

For queries that use disjunctions of predicates (i.e., OR), query performance can be improved by search optimization if ***all*** predicates adhere to the conditions above.

For example, suppose that a query has:

`where <condition_x> or <condition_y>`

Search optimization can improve performance if each condition separately returns a few rows (i.e., `<condition_x>` returns a few rows ***and*** `<condition_y>` returns a few rows).

If `<condition_x>` returns a few rows but `<condition_y>` returns many rows, the query performance ***does not*** benefit from search optimization.

In the case of disjunctions, each predicate in isolation is not decisive in the query. The other predicates need to be evaluated before it can be determined if search optimization can improve performance.

Views

The search optimization service can indirectly improve the performance of views (including secure views). If a base table for a view has search optimization enabled and if the query uses a selective predicate for that table, the search optimization service can improve performance when filtering rows.

All tables in the view do not need to have search optimization enabled. Search optimization is performed on each table independently.

Queries Not Supported By the Search Optimization Service

The search optimization service does not support the following:

- External tables.
- Materialized views.
- Columns defined with a COLLATE clause.
- Column concatenation.
- Analytical expressions.
- Tables and views protected by row access policies cannot be used with the Search Optimization Service.
- Casts on table columns (except for fixed-point numbers cast to strings).

  Although search optimization supports predicates with implicit and explicit casts on constant values, it does not support predicates that cast values in the actual table column (except for casts from INTEGER and NUMBER to VARCHAR).

  For example, the following predicates are supported because they use implicit and explicit casts on constant values (not values in the table column):

  ```
  -- Supported predicate
  -- (where the string '2020-01-01' is implicitly cast to a date)
  where timestamp1 = '2020-01-01';


  -- Supported predicate
  -- (where the string '2020-01-01' is explicitly cast to a date)
  where timestamp1 = '2020-01-01'::date;
  ```

  The following predicate is not supported because it uses a cast on values in the table column:

  ```
  -- Unsupported predicate
  -- (where values in a VARCHAR column are cast to DATE)
  where to_date(varchar_column) = '2020-01-01';
  ```

  The search optimization service considers the original column values, not the values after the cast. As a result, the search optimization service is not used for queries with these predicates.

As mentioned earlier, the exception to this rule is casting NUMBER or INTEGER values to VARCHAR values in the table column. The search optimization service does support this type of predicate:

```
-- Supported predicate
-- (where values in a numeric column are cast to a string)
where cast(numeric_column as varchar) = '2'
```

Search optimization does not improve performance of queries that use Time Travel because search optimization works only on active data.

The query acceleration service does not accelerate queries on tables that have search optimization service enabled.

Configuring Search Optimization for a Table

Note

Adding search optimization to a large table (a table containing terabytes (TB) or more of data) might result in an immediate increase in credit consumption over a short period of time.

Before you add search optimization to a large table, get an estimate of these costs so that you know what to expect.

To add search optimization to a table, you use the ALTER TABLE … ADD SEARCH OPTIMIZATION command. This sets up search access paths to improve the performance of equality and IN predicate queries for all columns that use the supported data types.

If you want to improve performance for other types of queries or if you need more control over which columns are configured for search optimization, use the ON clause in the ALTER TABLE … ADD SEARCH OPTIMIZATION command.

In the ON clause in ADD SEARCH OPTIMIZATION, you specify which columns should be enabled for search optimization. When enabling search optimization for a given column, you can also specify a query method (e.g. equality and IN searches, etc.).

To manage the cost of search optimization, you can remove search optimization from specific columns where search optimization is not needed.

Finally, as the search optimization service supports additional types of predicates and data types (e.g. substrings and regular expressions, GEOGRAPHY, fields in VARIANTs, etc.), you can specify which columns should be configured to take advantage of this support.

Preview Feature — Open

Configuring search optimization for specific columns is a preview feature that is available to all accounts that are Enterprise Edition (or higher).

This feature is being rolled out across different regions. For a list of the regions in which this feature is enabled, see Feature Rollout: Search Optimization Service Support for Column Configuration, String Patterns, VARIANT, and GEOGRAPHY.

Note: Adding search optimization for the entire table is a feature that is generally available.

The next sections explain how to configure search optimization for a table:

- Configuring Search Optimization for Specific Columns
- Adding Search Optimization for the Entire Table
- Verifying That the Table Is Configured For Search Optimization

Configuring Search Optimization for Specific Columns

To configure search optimization for a specific column, use the ALTER TABLE … ADD SEARCH OPTIMIZATION … command with the ON clause.

Preview Feature — Open

Configuring search optimization for specific columns is a preview feature that is available to all accounts that are Enterprise Edition (or higher).

This feature is being rolled out across different regions. For a list of the regions in which this feature is enabled, see Feature Rollout: Search Optimization Service Support for Column Configuration, String Patterns, VARIANT, and GEOGRAPHY.

Note

When running this command, use a role that has the privileges to add search optimization to the table.

The ON clause specifies that you want to configure search optimization for specific columns. For details on the syntax, see Search Optimization Actions (searchOptimizationAction).

Note

If you just want to apply search optimization for equality and IN predicates to all applicable columns in the table, see Adding Search Optimization for the Entire Table.

After running this command, you can verify that the columns have been configured for search optimization.

The next sections contain examples that demonstrate how to specify the configuration for search optimization:

- Example: Supporting Equality and IN Predicates for Specific Columns
- Example: Supporting Equality and IN Predicates for All Applicable Columns
- Example: Supporting Different Types of Predicates
- Example: Supporting Equality and IN Predicates for a Field in a VARIANT

Example: Supporting Equality and IN Predicates for Specific Columns

To optimize searches with equality predicates for the columns `c1`, `c2`, and `c3` in the table `t1`, execute the following statement:

You can also specify the same search method more than once in the ON clause:

```
-- This statement is equivalent to the previous statement.
alter table t1 add search optimization on equality(c1), equality(c2, c3);
```

Example: Supporting Equality and IN Predicates for All Applicable Columns

To optimize searches with equality predicates for all applicable columns in the table, execute the following statement:

```
alter table t1 add search optimization on equality(*);
```

Note the following:

- As explained in the description of the syntax for the search method and target, for a given method, you cannot specify an asterisk and specific columns.
- Although omitting the ON clause also configures search optimization for equality and IN predicates on all applicable columns in the table, there are differences between specifying and omitting the ON clause. See Adding Search Optimization for the Entire Table.

Example: Supporting Different Types of Predicates

To optimize searches with equality predicates for the column `c1` and `c2` and substring searches for the column `c3`, execute the following statement:

```
alter table t1 add search optimization on equality(c1, c2), substring(c3);
```

Example: Supporting Equality and IN Predicates for a Field in a VARIANT

To optimize searches with equality predicates on the VARIANT field `uuid` nested in the field `user` in the VARIANT column `c4`, execute the following statement:

```
alter table t1 add search optimization on equality(c4:user:uuid);
```

Adding Search Optimization for the Entire Table

If you just want to specify EQUALITY for all columns of the supported data types (except for VARIANT), use the ALTER TABLE … ADD SEARCH OPTIMIZATION command without the ON clause.

Note

When running this command, use a role that has the privileges to add search optimization to the table.

For example:

```
alter table test_table add search optimization;
```

For more information on the syntax, see the section on search optimization in ALTER TABLE.

After running this command, you can verify that the columns have been configured for search optimization.

Note the following:

- After you run this command, any columns that are subsequently added to the table will also be configured for EQUALITY.
- If you execute ALTER TABLE … { ADD | DROP } SEARCH OPTIMIZATION with the ON clause on the same table, any columns that are subsequently added to the table will not be configured for EQUALITY automatically.

  You must execute ALTER TABLE … ADD SEARCH OPTIMIZATION ON … to configure these newly added columns for EQUALITY.

Verifying That the Table Is Configured For Search Optimization

To verify that the table and its columns have been configured for search optimization:

For example:

```
show tables like '%test_table%';
```

In the output from this command:

- Verify that SEARCH_OPTIMIZATION is `ON`, which indicates that search optimization has been added.
- Check the value of SEARCH_OPTIMIZATION_PROGRESS. This specifies the percentage of the table that has been optimized so far.

  When search optimization is first added to a table, the performance benefits do not appear immediately. The search optimization service starts populating data in the background. The benefits appear increasingly as the maintenance catches up to the current state of the table.

  Before you run a query to verify that search optimization is working, wait until this shows that the table has been fully optimized.

3. Run a query to verify that search optimization is working.

   Note that the Snowflake optimizer automatically chooses when to use the search optimization service for a particular query. Users cannot control which queries search optimization is used for.

   Choose a query that the search optimization service is designed to optimize. See Identifying the Tables That Benefit From Search Optimization.

4. In the web UI, view the query plan for this query, and verify that the query node "Search Optimization Access" is part of the query plan.

Enabling the Search Optimization Service to Improve the Performance of Joins

Note

This feature is being rolled out across different regions. For a list of the regions in which this feature is enabled, see Feature Rollout: Support for Joins in the Search Optimization Service.

The search optimization service can improve the performance of queries that join a large table with one or more small tables (e.g. a fact table and several dimension tables).

For example, suppose that `product` is a table containing a row for each product, and `sales` is a table containing a row for each sale of a product. `product` contains fewer rows and is smaller than `sales`. To find all sales of a specific product, you join the `sales` table (the larger table) with the `product` table (the smaller table). The search optimization service can improve the performance of this type of join.

Note

In data warehousing, the large table is often referred to as the fact table. The small table is referred to as the dimension table. The rest of this topic uses these terms when referring to the large table and the small table in the join.

To enable the search optimization service to improve the performance of joins, add search optimization to the fact table (the larger of the two tables).

To take advantage of search optimization, the dimension table (the smaller of the two tables) should have few distinct values. The search optimization costs of a query are proportionate to the number of distinct values that need to be looked up in the fact table. If the number of distinct values in the dimension table is too large, Snowflake might decide against using the search access path and use the regular table access path instead.

Supported Join Predicates

The search optimization service can improve the performance of queries with the following types of join predicates:

- Equality predicates of the form `<dimension_table>.<column> = <fact_table>.<column>`.
- Transformations on the side of the predicate with the dimension (e.g. string concatenation, addition, etc.).
- Conjunctions (`AND`) of multiple equality predicates.

Example: Simple Equality Predicate as the Join Predicate

The following is an example of a supported query that uses a simple equality predicate as the join predicate. This query joins a fact table named `sales` with a dimension table named `product`. The fact table is large and has search optimization enabled. The input from the dimension table is small, due to the selective filter on the `category` column.

```
select sales.date, product.name
from sales join product on (sales.product_id = product.id)
where product.category = 'Cutlery';
```

Example: Join Predicate Transformed on the Side with the Dimension

Queries that transform the side of the predicate with the dimension (e.g multiplies) can also benefit from search optimization:

```
select sales.date, product.name
from sales join product on (sales.product_id = product.old_id * 100)
where product.category = 'Cutlery';
```

Example: Join Predicate Spanning Multiple Columns

Queries in which a join predicate spans multiple columns are supported as well:

```
select sales.date, product.name
from sales join product on (sales.product_id = product.id and sales.location = product.place_of_production)
where product.category = 'Cutlery';
```

Example: Query Using Point-Lookup Filters and Join Predicates

In a query that uses both regular point-lookup filters and join predicates, the search optimization service can improve the performance of both. In the following query, the search optimization service improves the `sales.location` point-lookup predicate as well as the `product_id` join predicate.

```
select sales.date, product.name
from sales join product on (sales.product_id = product.id)
where product.category = 'Cutlery'
and sales.location = 'Buenos Aires';
```

Limitations in the Support for Joins

- Disjuncts (`OR`) in join predicates are currently not supported.
- LIKE/ILIKE/RLIKE join predicates are currently not supported.
- Join predicates on variant columns are currently not supported.
- EQUAL_NULL equality predicates will not be supported.
- The current limitations of the search optimization service also apply to this feature.

Displaying the Search Optimization Configuration for a Table

To display the search optimization configuration for a table, use the DESCRIBE SEARCH OPTIMIZATION command.

For example, suppose that you execute the following statement to configure search optimization for a column:

```
alter table t1 add search optimization on equality(c1);
```

Executing DESCRIBE SEARCH OPTIMIZATION produces the following output:

Preview Feature — Open

Displaying the search optimization configuration is part of a preview feature that is available to all accounts that are Enterprise Edition (or higher).

This feature is being rolled out across different regions. For a list of the regions in which this feature is enabled, see Feature Rollout: Search Optimization Service Support for Column Configuration, String Patterns, VARIANT, and GEOGRAPHY.

```
| expression_id |  method   | target | target_data_type | active |
+---------------+-----------+--------+------------------+--------+
| 1             | EQUALITY  | C1     | NUMBER(38,0)     | true   |
+---------------+-----------+--------+------------------+--------+
```

Working With Tables Optimized For Search

When working with a table that uses search optimization, you need to be aware of the effects on the search optimization service.

Modifying the Table

A search access path becomes invalid if the default value of a column is changed.

To use search optimization again after a search access path has become invalid, you must drop and add back the search optimization property for the table.

A search access path remains valid if you add, drop, or rename a column:

- When you add a column to a table, the new column is added to the search access path automatically.
- When you drop a column from a table, the dropped column is removed from the search access path automatically.
- Renaming a column doesn't require any changes to the search access path.

If you drop a table, the search optimization property and search access paths are also dropped. Note that:

- Undropping the table immediately reestablishes search optimization as a property of the table.
- When you drop a table, the search access path has the same data retention period as the table.

If you drop the search optimization property from the table, the search access path is removed. When you add the property back, the maintenance service needs to recreate the search access path. (There is no way to undrop the property.)

Cloning the Table, Schema, or Database

If you clone a table, schema, or database, the search optimization property and search access paths of each table are also cloned. (Cloning a table, schema, or database creates a zero-copy clone of each table and its corresponding search access paths.)

Note that if you use CREATE TABLE … LIKE to create a new empty table with the same columns as the original table, the search optimization property is not copied to the new table.

Working With Tables in a Secondary Database (Database Replication Support)

If a table in the primary database has the search optimization property enabled, the property is replicated to the corresponding table in the secondary database.

Search access paths in the secondary database are not replicated but are instead rebuilt automatically. Note that this process incurs the same kinds of costs described in Managing the Costs of the Search Optimization Service.

Sharing the Table

Data providers can use Secure Data Sharing to share tables that have search optimization enabled.

When querying shared tables, data consumers can benefit from any performance improvements made by the search optimization service.

Managing the Costs of the Search Optimization Service

The search optimization service impacts costs for both storage and compute resources:

- Storage resources: The search optimization service creates a search access path data structure that requires space for each table on which search optimization is enabled. The storage cost of the search access path depends upon multiple factors, including:
  - The number of distinct values (NDVs) in the table. In the extreme case where all columns have data types that use the search access path, and all data values in each column are unique, the required storage can be as much as the original table's size.

    Typically, however, the size is approximately 1/4 of the original table's size.
- Compute resources:

Automatic clustering, while improving the latency of queries in tables with search optimization, can further increase the maintenance costs of search optimization. If a table has a high churn rate, enabling automatic clustering and configuring search optimization for the table can result in higher maintenance costs than if the table is just configured for search optimization.

Snowflake ensures efficient credit usage by billing your account only for the actual resources used. Billing is calculated in 1-second increments.

See the “Serverless Feature Credit Table” in the Snowflake service consumption table for the costs per compute hour.

Once you enable the search optimization service, you can view the costs for your use of the service.

Tip

Snowflake recommends starting slowly with this feature (i.e. adding search optimization to only a few tables at first) and closely monitoring the costs and benefits.

Estimating the Costs

To estimate the cost of adding search optimization to a table and configuring specific columns for search optimization, use the SYSTEM$ESTIMATE_SEARCH_OPTIMIZATION_COSTS function.

In general, the costs are proportional to:

- The number of tables on which the feature is enabled, and the number of distinct values in those tables.
- The amount of data that changes in these tables.

Viewing the Costs

You can view the billing costs for the search optimization service by using either the web interface or SQL. See Understanding Billing for Serverless Features.

Reducing the Costs

You can control the cost of the search optimization service by carefully choosing the tables for which to enable search optimization.

In addition, to reduce the cost of the search optimization service:

- Snowflake recommends batching DML operations on the table:
  - `DELETE`: If tables store data for the most recent time period (e.g. the most recent day or week or month), then when you trim your table by deleting old data, the search optimization service must take into account the updates. In some cases, you might be able to reduce costs by deleting less frequently (e.g. daily rather than hourly).
  - `INSERT`, `UPDATE`, and `MERGE`: Batching these types of DML statements on the table can reduce the cost of maintenance by the search optimization service.
- If you recluster the entire table, consider dropping the search optimization property for that table before reclustering, and then re-adding the search optimization service after reclustering.

Removing the Search Optimization Property From a Table

You can remove the search optimization configuration for specific columns or for the entire table.

- Dropping Search Optimization for Specific Columns
- Removing the Search Optimization From the Table

Dropping Search Optimization for Specific Columns

To drop the search optimization configuration for specific columns, use the following command: ALTER TABLE … ADD SEARCH OPTIMIZATION … command with the ON clause.

For example, suppose that executing the DESCRIBE SEARCH OPTIMIZATION command prints the following expressions:

Preview Feature — Open

Dropping the search optimization configuration for specific columns is a preview feature that is available to all accounts that are Enterprise Edition (or higher).

This feature is being rolled out across different regions. For a list of the regions in which this feature is enabled, see Feature Rollout: Search

```
| expression_id | method    | target    | target_data_type  | active |
+---------------+-----------+-----------+-------------------+--------+
|             1 | EQUALITY  | C1        | NUMBER(38,0)      | true   |
|             2 | EQUALITY  | C2        | VARCHAR(16777216) | true   |
|             3 | EQUALITY  | C4        | NUMBER(38,0)      | true   |
|             4 | EQUALITY  | C5        | VARCHAR(16777216) | true   |
|             5 | EQUALITY  | V1        | VARIANT           | true   |
|             6 | SUBSTRING | C2        | VARCHAR(16777216) | true   |
|             7 | SUBSTRING | C5        | VARCHAR(16777216) | true   |
|             8 | GEO       | G1        | GEOGRAPHY         | true   |
|             9 | EQUALITY  | V1:"key1" | VARIANT           | true   |
|            10 | EQUALITY  | V1:"key2" | VARIANT           | true   |
+---------------+-----------+-----------+-------------------+--------+
```

To drop search optimization for substrings on the column `c2`, execute the following statement:

```
alter table t1 drop search optimization on substring(c2);
```

To drop all search optimization for all methods on the column `c5`, execute the following statement:

```
alter table t1 drop search optimization on c5;
```

Because the column `c5` is configured to optimize equality and substring searches, the statement above drops the configuration for equality and substring searches for `c5`.

To drop search optimization for equality on the column `c1` and to drop the configuration specified by the expression IDs `6` and `8`, execute the following statement:

```
alter table t1 drop search optimization on equality(c1), 6, 8;
```

Removing the Search Optimization From the Table

To remove the search optimization property from a table:

1. Switch to a role that has the privileges to remove search optimization from the table.
2. Run the following command:

   ```
   ALTER TABLE [IF EXISTS] <table_name> DROP SEARCH OPTIMIZATION;
   ```

   For example:

   ```
   alter table test_table drop search optimization;
   ```

   For more information, see the section on search optimization in ALTER TABLE.

Examples

The following code shows creation and use of a search optimization service.

Start by creating a table with data:

```
create or replace table test_table (id int, c1 int, c2 string, c3 date) as
select * from values
  (1, 3, '4',  '1985-05-11'),
  (2, 4, '3',  '1996-12-20'),
  (3, 2, '1',  '1974-02-03'),
  (4, 1, '2',  '2004-03-09'),
  (5, null, null,  null);
```

Add the search optimization property to the table:

The following queries can use the search optimization service:

```
select * from test_table where id = 2;
```

```
select * from test_table where c2 = '1';
```

```
select * from test_table where c3 = '1985-05-11';
```

```
select * from test_table where c1 is null;
```

```
select * from test_table where c1 = 4 and c3 = '1996-12-20';
```

The following query can use the search optimization because the implicit cast is on the constant, not the column:

```
select * from test_table where c2 = 2;
```

The following ***cannot*** use the search optimization because the cast is on the table's column:

```
select * from test_table where cast(c2 as number) = 2;
```

An IN clause is compatible with search optimization:

```
select id, c1, c2, c3
    from test_table
    where id in (2, 3)
    order by id;
```

If predicates are individually compatible with search optimization, then they can be joined by the conjunction `AND` and still be compatible with search optimization:

```
select id, c1, c2, c3
    from test_table
    where c1 = 1
       and
          c3 = to_date('2004-03-09')
    order by id;
```

DELETE and UPDATE (and MERGE) can also use the search optimization service:

```
delete from test_table where id = 3;
```

```
update test_table set c1 = 99 where id = 4;
```

PREVIOUS | NEXT

ASK THE COMMUNITY CONTACT SUPPORT REPORT DOC ISSUE

snowflake®

Cloud Data Platform
Architecture
Pricing
Data Marketplace

Snowflake for Healthcare & Life Sciences
Snowflake for Marketing Analytics
Snowflake for Retail
Snowflake for Education
Snowflake for Developers

Resource Library
Webinars
Community
Legal

EXPLORE
News
Blog
Trending

ABOUT
About Snowflake
Leadership & Board
Careers
Contact

Privacy Notice | Site Terms

© 2022 Snowflake Inc. All Rights Reserved