save-classic.md

---

id: save-classic

title: save (Classic) Search Operator

sidebar_label: save (Classic)

The `save` (classic) operator works with the classic Lookup Tables feature.

:::info

For information about our newer, more scalable Lookup Tables feature and the new `save` operator that works with it, see [Lookup Tables](/docs/search/lookup-tables) and [`save`](save.md). The new `save` operator allows you to merge new and changed rows, whereas the `save` (classic) operator can only append to existing rows.

:::

Using the `save` (classic) operator allows you to save the results of a query into the Sumo Logic file system. Later, you can use the lookup operator to access the saved data. The save operator saves data in a simple format to a location you choose.

You will need to remember the path where you point the save operator to put the file. You may want to save searches that contain save operators so you can refer to it later. There is no way to locate the saved file if you forget the path.

Syntax

```sql

save [append] <myFolder/mySubFolder/myFileName>

```

Rules

* You can successfully query a lookup as long as the file size is less than 8MB, and you can save up to 500MB of data. If the file size exceeds 8MB, you will get the **File Too Big** error.

* Queries that use the Save operator cannot be [pinned](/docs/get-started/library#search-the-library).

Examples

Let's say you want to save data about new user accounts created each day. Your Save operator could look like:

```sql

| parse "name=*," as name

| parse "action=*," as action

| parse "date=*," as date

| where action="sign-up"

| first(date) as date, first(action) as action by name

| save myFolder/mySubFolder/newDailyUsers

```

The above search would create a file that looks like this:

| Name | Action | Date |

|:-- |:-- |:-- |

| John | sign-up | 2012-08-20 |

| Bill | sign-up | 2012-08-21 |

| Bob | sign-up | 2012-08-21 |

You can access data in the saved table using the lookup operator.

Aggregate results can also be saved with the save operator.

Use the Fields operator to remove unnecessary fields

When creating a save file, make sure that the file is as small as possible to work more quickly. A good way to do this is to remove unnecessary fields using the [`fields`](fields.md) operator. This includes [built-in metadata fields](/docs/search/get-started-with-search/search-basics/built-in-metadata), like `_raw`.

Saving files to a shared location

A file generated by a save operator can be saved to an org-level shared folder. This allows for others in your organization to use your search results when running their lookup queries.
:::note

Files saved to a shared location can only be modified by the person who originally shared the file.

:::

To save a file to a shared location include the word **shared** at the beginning:

```sql

...save /shared/myFolder/mySubFolder/fileName

```

For more information, see [Using Lookup to Access Saved Data](lookup-classic.md).

Appending to saved files

Once you've created a file generated by a save operator, you can append data at any time. If you are running a scheduled daily search that calculates properties for the current day, that data is appended to the existing file containing results from the previous days. Data you append to a file must match exactly; if the new results do not match the previous results an error message appears, including cases where you attempt to append with additional fields.

If you do not use "append" the previously saved data will be overwritten.

Let's say that you'd like to append to your **newDailyUsers** file each day by scheduling this search to run every 24 hours:

```sql
| parse "name=*," as name

| parse "action=*," as action

| parse "date=*," as date

| where action="sign-up"

| first(date) as date, first(action) as action by name

| save append myFolder/mySubFolder/newDailyUsers
```

Each day the query runs the above data is appended to the **newDailyUsers** file.

You can also append data to a saved file from different queries. For example, say we have two sources, "bill" that includes billing information, and "config" that contains account information, and we'd like to be able to search for some values from each source. These searches would create a table with information from both sources:

```sql
_source=bill | parse "user_id=*," as name

| parse "user_email=*," as email

| save myFolder/mySubFolder/NameEmailMapping
```

```sql
_source=config | parse "_user=[*]" as name

| parse "contact_info=[*]" as email

| save append myFolder/mySubFolder/NameEmailMapping
```

```
```