

Session 3-2: Diving into Data – Search and Report for Admins

Table of Contents

Session 3-2: Diving into Data – Search and Report for Admins	4
Introduction	4
TIP	5
TIP	6
NOTE	8
Making raw event data readable.....	10
Getting ready.....	10
How to do it... ..	10
How it works... ..	12
TIP	13
There's more... ..	13
TABULATING EVERY FIELD	13
REMOVING FIELDS, THEN TABULATING EVERYTHING ELSE	13
TIP	14
Finding the most accessed web pages.....	15
Getting ready.....	15
How to do it... ..	15
How it works... ..	16
There's more... ..	17
SEARCHING FOR THE TOP 10 ACCESSED WEB PAGES.....	17
SEARCHING FOR THE MOST ACCESSED PAGES BY USER.....	17
NOTE	18
Finding the most used web browsers.....	19
Getting ready.....	19
How to do it... ..	19
How it works... ..	20
There's more... ..	21
SEARCHING FOR THE WEB BROWSER DATA FOR THE MOST USED OS TYPES	21
Identifying the top-referring websites	22
Getting ready.....	22
How to do it... ..	22

How it works.....	23
There's more.....	23
SEARCHING FOR THE TOP 10 USING STATS INSTEAD OF TOP	23
NOTE	24
Charting web page response codes.....	25
Getting ready.....	25
How to do it... ..	25
How it works.....	26
There's more.....	27
TOTALING SUCCESS AND ERROR WEB PAGE RESPONSE CODES.....	27
Displaying web page response time statistics	29
Getting ready.....	29
How to do it... ..	29
How it works.....	30
There's more.....	31
DISPLAYING WEB PAGE RESPONSE TIME BY ACTION	31
Listing the top viewed products.....	33
Getting ready.....	33
How to do it... ..	33
How it works.....	34
There's more.....	35
SEARCHING FOR THE PERCENTAGE OF CART ADDITIONS FROM PRODUCT VIEWS	35
Charting the application's functional performance	37
Getting ready.....	37
How to do it.....	37
How it works.....	38
TIP	39
There's more.....	39
Charting the application's memory usage	40
Getting ready.....	40
How to do it... ..	40
How it works.....	41

Counting the total number of database connections.....	43
Getting ready.....	43
How to do it.....	43
How it works.....	44

Session 3-2: Diving into Data – Search and Report for Admins

In this session, we will cover the basic ways to search data in Splunk. We will cover the following recipes:

- Making raw event data readable
- Finding the most accessed web pages
- Finding the most used web browsers
- Identifying the top-referring websites
- Charting web page response codes
- Displaying web page response time statistics
- Listing the top-viewed products
- Charting the application's functional performance
- Charting the application's memory usage
- Counting the total number of database connections

Introduction

In the previous session, we learned about the various ways to get data into Splunk. In this session, we will dive right into the data and get our hands dirty.

The ability to search machine data is one of Splunk's core functions, and it should come as no surprise that many other features and functions of Splunk are heavily driven-off searches. Everything from basic reports and dashboards to data models and fully featured Splunk applications are powered by Splunk searches behind the scenes.

Splunk has its own search language known as the **Search Processing Language (SPL)**. This SPL contains hundreds of search commands, most of which also have several functions, arguments, and clauses. While a basic understanding of SPL is required in order to effectively search your data in Splunk, you are not expected to know all the commands! Even the most seasoned ninjas do not know all

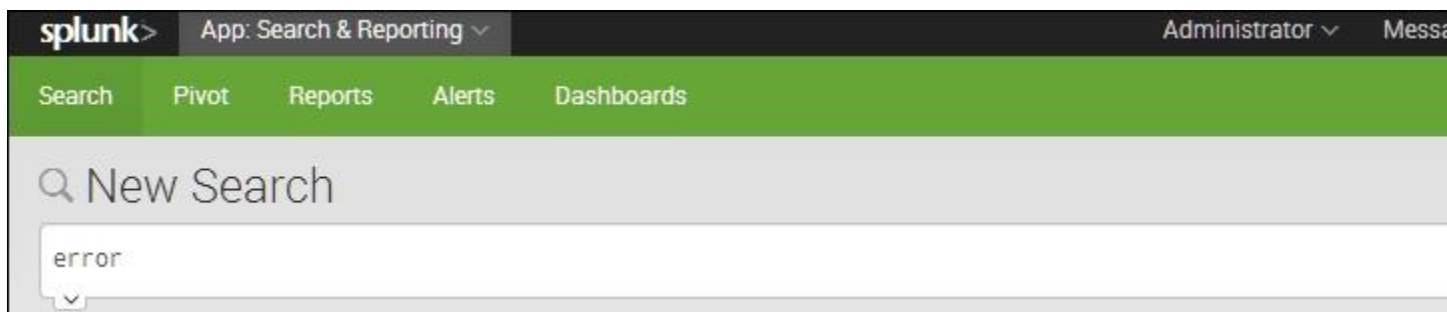
the commands and regularly refer to the Splunk manuals, website, or Splunk Answers (<http://answers.splunk.com>).

TIP

To get you on your way with SPL, be sure to check out the search command cheat sheet and download the handy quick reference guide available at <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/SplunkEnterpriseQuickReferenceGuide>.

Searching

Searches in Splunk usually start with a base search, followed by a number of commands that are delimited by one or more pipe (|) characters. The result of a command or search to the left of the pipe is used as the input for the next command to the right of the pipe. Multiple pipes are often found in a Splunk search to continually refine data results as needed. As we go through this session, this concept will become very familiar to you.



Splunk allows you to search for anything that might be found in your log data. For example, the most basic search in Splunk might be a search for a keyword such as `error` or an IP address such as `10.10.12.150`. However, searching for a single word or IP over the terabytes of data that might potentially be in Splunk is not very efficient. Therefore, we can use the SPL and a number of Splunk commands to really refine our searches. The more refined and granular the search, the faster the time to run and the quicker you get to the data you are looking for!

TIP

When searching in Splunk, try to filter as much as possible before the first pipe (|) character, as this will save CPU and disk I/O. Also, pick your time range wisely. Often, it helps to run the search over a small time range when testing it and then extend the range once the search provides what you need.

Boolean operators

There are three different types of Boolean operators available in Splunk. These are `AND`, `OR`, and `NOT`. Case sensitivity is important here, and these operators must be in uppercase to be recognized by Splunk. The `AND` operator is implied by default and is not needed, but does no harm if used.

For example, searching for the term `error` or `success` would return all the events that contain either the word `error` or the word `success`. Searching for `error success` would return all the events that contain the words `error` and `success`. Another way to write this can be `error AND success`. Searching web access logs for `error OR success NOT mozilla` would return all the events that contain either the word `error` or `success`, but not those events that also contain the word `mozilla`.

Common commands

There are many commands in Splunk that you will likely use on a daily basis when searching data within Splunk. These common commands are outlined in the following table:

Command	Description
<code>chart/timechart</code>	This command outputs results in a tabular and/or time-based output for use by Splunk charts.
<code>dedup</code>	This command de-duplicates results based upon specified fields, keeping the most recent match.
<code>eval</code>	This command evaluates new or existing fields and values. There are many different functions available for <code>eval</code> .
<code>fields</code>	This command specifies the fields to keep or remove in search results.

Command	Description
<code>head</code>	This command keeps the first <code>x</code> (as specified) rows of results.
<code>lookup</code>	This command looks up fields against an external source or list, to return additional field values.
<code>rare</code>	This command identifies the least common values of a field.
<code>rename</code>	This command renames the fields.
<code>replace</code>	This command replaces the values of fields with another value.
<code>search</code>	This command permits subsequent searching and filtering of results.
<code>sort</code>	This command sorts results in either ascending or descending order.
<code>stats</code>	This command performs statistical operations on the results. There are many different functions available for <code>stats</code> .
<code>table</code>	This command formats the results into a tabular output.
<code>tail</code>	This command keeps only the last <code>x</code> (as specified) rows of results.
<code>top</code>	This command identifies the most common values of a field.
<code>transaction</code>	This command merges events into a single event based upon a common transaction identifier.

Time modifiers

The drop-down time range picker in the **Graphical User Interface (GUI)** to the right of the Splunk search bar allows users to select from a number of different preset and custom time ranges. However, in addition to using the GUI, you can also specify time ranges directly in your search string using the earliest and latest time modifiers. When a time modifier is used in this way, it automatically overrides any time range that might be set in the GUI time range picker.

The `earliest` and `latest` time modifiers can accept a number of different time units: seconds (`s`), minutes (`m`), hours (`h`), days (`d`), weeks (`w`), months (`mon`), quarters (`q`), and years (`y`). Time modifiers can also make use of the `@` symbol to round down and snap to a specified time.

For example, searching for `sourcetype=access_combined earliest=-1d@d latest=-1h` will search all the `access_combined` events from midnight a day ago until an hour ago from now. Note that the snap (`@`) will round down such that if it were 12 p.m. now, we would be searching from midnight a day and a half ago until 11 a.m. today.

Working with fields

Fields in Splunk can be thought of as keywords that have one or more values. These fields are fully searchable by Splunk. At a minimum, every data source that comes into Splunk will have the `source`, `host`, `index`, and `sourcetype` fields, but some source might have hundreds of additional fields. If the raw log data contains key-value pairs or is in a structured format such as JSON or XML, then Splunk will automatically extract the fields and make them searchable. Splunk can also be told how to extract fields from the raw log data in the backend `props.conf` and `transforms.conf` configuration files.

Searching for specific field values is simple. For example, `sourcetype=access_combined status!=200` will search for events with a `sourcetype` field value of `access_combined` that has a status field with a value other than `200`.

NOTE

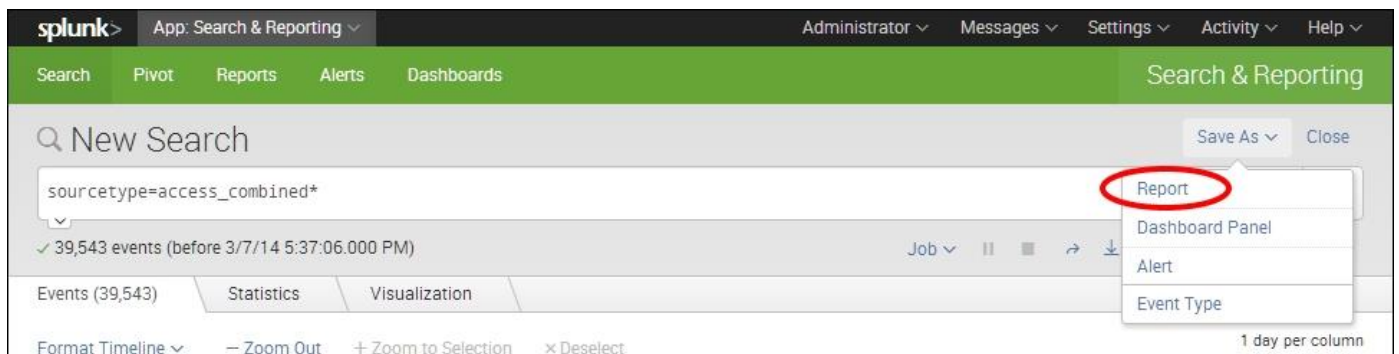
Splunk has a number of built-in pre-trained `sourcetypes` that ship with Splunk Enterprise that might work with out-of-the-box, common data sources. These are available at <http://docs.splunk.com/Documentation/Splunk/latest/Data/Listofpretrainedsourcetypes>.

In addition, **Technical Add-Ons (TAs)**, which contain event types and field extractions for many other common data sources such as Windows events, are available from the Splunk app store at <https://splunkbase.splunk.com>.

Saving searches

Once you have written a nice search in Splunk, you may wish to save the search so that you can use it again at a later date or use it for a dashboard. Saved searches in Splunk are known as **Reports**. To save a

search in Splunk, you simply click on the **Save As** button on the top right-hand side of the main search bar and select **Report**.



Making raw event data readable

When a basic search is executed in Splunk from the search bar, the search results are displayed in a raw event format by default. To many users, this raw event information is not particularly readable, and valuable information is often clouded by other less valuable data within the event. Additionally, if the events span several lines, only a few events can be seen on the screen at any one time.

In this recipe, we will write a Splunk search to demonstrate how we can leverage Splunk commands to make raw event data readable, tabulating events and displaying only the fields we are interested in.

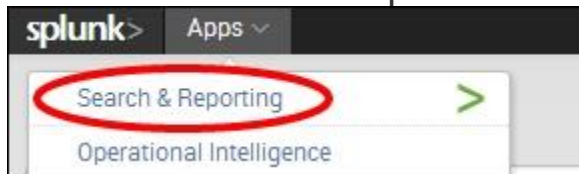
Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and search results area.

How to do it...

Follow the given steps to search and tabulate the selected event data:

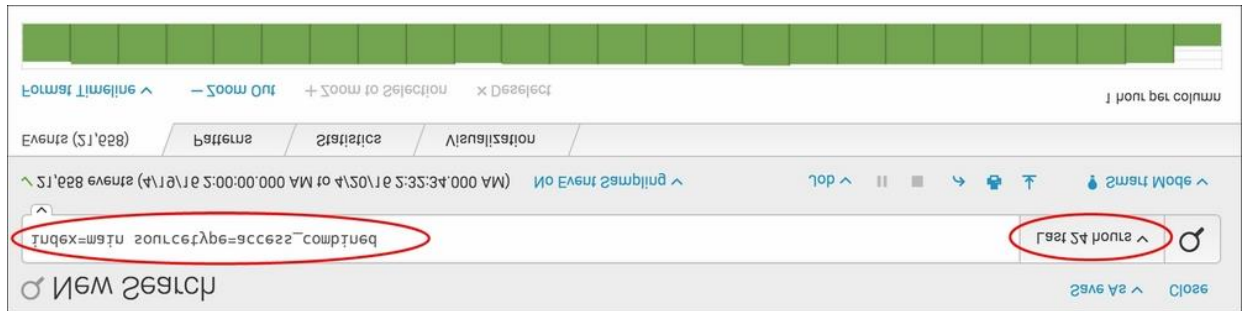
1. Log in to your Splunk server.
2. Select the **Search & Reporting** application from the drop-down menu located in the top left-hand side of the screen.



3. Set the time range picker to **Last 24 hours** and type the following search into the Splunk search bar:

```
index=main sourcetype=access_combined
```

Then, click on **Search** or hit *Enter*.



4. Splunk will return the results of the search and display the raw search events under the search bar.
5. Let's rerun the search, but this time we will add the `table` command as follows:

```
index=main sourcetype=access_combined | table _time,
referer_domain, method, uri_path, status, JSESSIONID, useragent
```

6. Splunk will now return the same number of events, but instead of presenting the raw events to you, the data will be in a nicely formatted table, displaying only the fields we specified. This is much easier to read!

New Search

index=main sourcetype=access_combined | table _time, referer_domain, method, uri_path, status, JSESSIONID, useragent

Last 24 hours

21,699 events (4/19/16 2:00:00.000 AM to 4/20/16 2:35:26.000 AM) No Event Sampling

Job

Smart Mode

Events

Patterns

Statistics (21,699)

Visualization

20 Per Page

Format

Preview

Prev

1

2

3

4

5

6

7

8

9

...

Next

_time	referer_domain	method	uri_path	status	JSESSIONID	useragent
2016-04-20 02:35:20	https://www1.samplesite.ca	GET	/viewCart	200	3EC32502F7653FA4FE3745FCC8043429	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0
2016-04-20 02:35:18	https://www1.samplesite.ca	POST	/removeItem	200	3EC32502F7653FA4FE3745FCC8043429	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0
2016-04-20 02:35:12	https://www1.samplesite.ca	GET	/viewCart	200	3EC32502F7653FA4FE3745FCC8043429	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0
2016-04-20 02:35:08	https://www1.samplesite.ca	POST	/addItem	200	3EC32502F7653FA4FE3745FCC8043429	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0

7. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_tabulated_webaccess_logs` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

Save As Report

Title: cp02_tabulated_webaccess_logs

Description: optional

Visualization: Single Value

Time Range Picker: Yes

Buttons: Cancel, Save

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main</code>	All the data in Splunk is held in one or more indexes. While not strictly necessary, it is a good practice to specify the index (<i>es</i>) to search, as this will ensure a more precise search.
<code>sourcetype=access_combined</code>	This tells Splunk to search only the data associated with the <code>access_combined</code> sourcetype, which, in our case, is the web access logs.
<code> table _time, referer_domain, method, uri_path, action, JSESSIONID, useragent</code>	Using the <code>table</code> command, we take the result of our search to the left of the pipe and tell Splunk to return the data in a tabular format. Splunk will only display the fields specified after the table command in the table of results.

In this recipe, you used the `table` command. The `table` command can have a noticeable performance impact on large searches. It should be used towards the end of a search, once all the other processing on the data by the other Splunk commands has been performed.

TIP

The `stats` command is more efficient than the `table` command and should be used in place of `table` where possible. However, be aware that `stats` and `table` are two very different commands.

There's more...

The `table` command is very useful in situations where we wish to present data in a readable format. Additionally, tabulated data in Splunk can be downloaded as a CSV file, which many users find useful for offline processing in spreadsheet software or for sending to others. There are some other ways we can leverage the `table` command to make our raw event data readable.

TABULATING EVERY FIELD

Often, there are situations where we want to present every event within the data in a tabular format, without having to specify each field one by one. To do this, we simply use a wildcard (*) character as follows:

```
index=main sourcetype=access_combined | table *
```

REMOVING FIELDS, THEN TABULATING EVERYTHING ELSE

While tabulating every field using the wildcard (*) character is useful, you will notice that there are a number of Splunk internal fields, such as `_raw`, that appear in the table. We can use the `fields` command before the `table` command to remove the fields as follows:

```
index=main sourcetype=access_combined | fields - sourcetype,
index, _raw, source date* linecount punct host time*
eventtype | table *
```

If we do not include the minus (-) character after the `fields` command, Splunk will keep the specified fields and remove all the other fields.

TIP

If you regularly need to remove a number of fields in your searches, you can write a macro to do this and then simply call the macro from your search. Macros are covered later in this book.

Finding the most accessed web pages

One of the data samples we loaded in Session 3-1, *Play Time – Getting Data In*, contained access logs from our web server. These have a Splunk sourcetype of `access_combined` and detail all pages accessed by the users of our web application. We are particularly interested in knowing which pages are being accessed the most, as this information provides great insight into how our e-commerce web application is being used. It could also help influence changes to our web application such that rarely visited pages are removed, or our application is redesigned to be more efficient.

In this recipe, we will write a Splunk search to find the most accessed web pages over a given period of time.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from Session 3-1, *Play Time – Getting Data In*. You should be familiar with the Splunk search bar and the time range picker to the right of it.

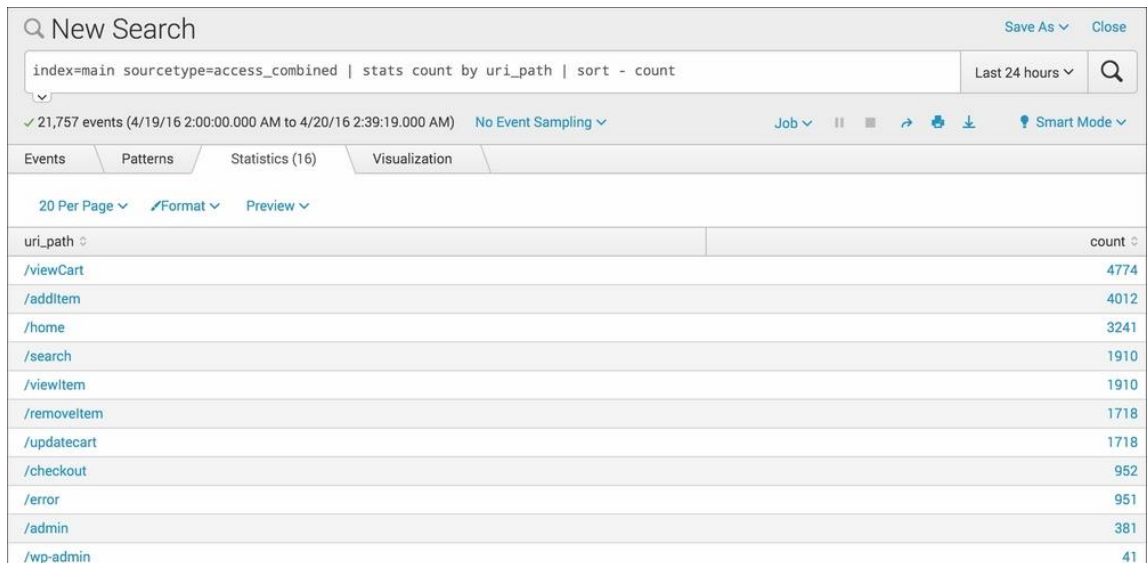
How to do it...

Follow the given steps to search for the most accessed web pages:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Set the range picker to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | stats count by uri_path  
| sort - count
```


4. Splunk will return a list of pages, and a new field named **count** displays the total number of times a page has been accessed.



The screenshot shows the Splunk search interface. At the top, there's a search bar with the query `index=main sourcetype=access_combined | stats count by uri_path | sort - count`. Below the search bar, it indicates 21,757 events from 4/19/16 2:00:00 AM to 4/20/16 2:39:19 AM. The interface has tabs for Events, Patterns, Statistics (16), and Visualization. The Statistics tab is active, showing a table with two columns: uri_path and count. The table lists various web pages and their access counts.

uri_path	count
/viewCart	4774
/addItem	4012
/home	3241
/search	1910
/viewItem	1910
/removeItem	1718
/updatecart	1718
/checkout	952
/error	951
/admin	381
/wp-admin	41

5. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_most_accessed_webpages` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main</code>	All the data in Splunk is held in one or more indexes. While not strictly necessary, it is a good practice to specify the index(es) to search, as this will ensure a more precise search.
<code>sourcetype=access_combined</code>	This tells Splunk to search only the data associated with the <code>access_combined</code> sourcetype, which, in our case, is the web access logs.
<code> stats count by uri_path</code>	Using the <code>stats</code> command, we take the result of our search to the left-hand side of the pipe and tell Splunk to count the instances of each <code>uri_path</code> . The <code>uri_path</code> field is the name of the field associated with the website page.

Search fragment	Description
<code> sort - count</code>	Using the <code>sort</code> command, we take the <code>count</code> field generated by <code>stats</code> and tell Splunk to sort the results of the previous command in descending (-) order, such that the most visited web page appears at the top of the results.

There's more...

We can further build upon the base search to provide different variations of the results.

SEARCHING FOR THE TOP 10 ACCESSED WEB PAGES

We can modify the search from this recipe and replace the `stats` command with the `top` command. By default, this will display the top 10 web pages:

```
sourcetype=access_combined index=main | top uri_path
```

Here, we modified the search and replaced the `stats` command with the `top` command. By default, this displays the top 10 web pages. If we want to get the top 20 web pages, we can specify a limit value, as follows:

```
sourcetype=access_combined index=main | top limit=20 uri_path
```

SEARCHING FOR THE MOST ACCESSED PAGES BY USER

We can modify the search from this recipe and can use the distinct count (`dc`) function of the `stats` command to display a list of users and the unique pages they visited:

```
sourcetype=access_combined index=main | stats dc(uri_path) by user | sort - user
```

The distinct count function ensures that if a user visits the same page multiple times, it is only counted as one visit. The user who visited the most number of unique pages will be at the top of the list, as we used a descending sort.

NOTE

For more information on the various functions that can be used with the `stats` command, check out <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/CommonStatsFunctions>.

Finding the most used web browsers

Users visiting our website use a variety of devices and web browsers. By analyzing the web access logs, we can understand which browsers are the most popular and, therefore, which browsers our site must support at the least. We can also use this same information to help identify the types of devices that people are using.

In this recipe, we will write a Splunk search to find the most used web browsers over a given period of time. We will then make use of both the `eval` and `replace` commands to clean up the data a bit.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from **Session 3-1, *Play Time – Getting Data In***. You should be familiar with the Splunk search bar and the time range picker to the right of it.

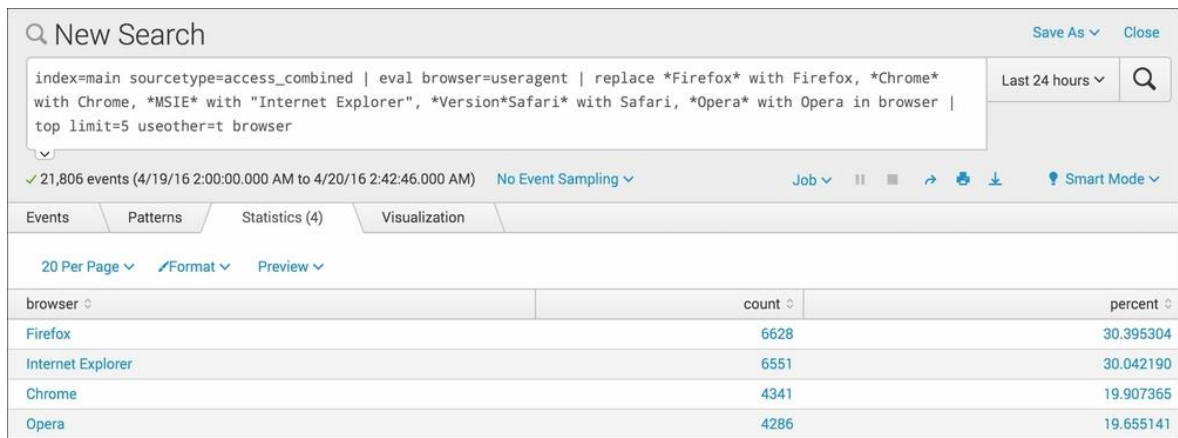
How to do it...

Follow the given steps to search for the most used web browsers:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | eval browser=useragent  
| replace *Firefox* with Firefox, *Chrome* with Chrome, *MSIE*  
with "Internet Explorer", *Version*Safari* with Safari, *Opera*  
with Opera in browser | top limit=5 useother=t browser
```

4. Splunk will return a tabulated list of the top five most used web browsers on our site, by count and percent.



5. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_most_used_webrowsers` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main</code> <code>sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this session.
<code> eval browser=useragent</code>	Using the <code>eval</code> command, we evaluate a new field called <code>browser</code> and populate it with the contents of the <code>useragent</code> field.
<code> replace *Firefox* with Firefox,</code> <code>*Chrome* with Chrome, *MSIE* with</code> <code>"Internet Explorer",</code> <code>*Version*Safari* with Safari,</code> <code>*Opera* with Opera in browser</code>	Using the <code>replace</code> command, we use wildcards (*) within the content of the <code>browser</code> field to replace the values with shortened browser names. Note that the values that contain spaces require quotes around them, for example, <code>"Internet Explorer"</code> .
<code> top limit=5 useother=t browser</code>	Using the <code>top</code> command, we tell Splunk to find the top five web browsers and classify everything else under the value of <code>other</code> .

In this recipe, we used both the `eval` and `replace` commands for illustrative purposes. This approach absolutely works, but a better approach can be to use Splunk's lookup functionality to look up the `useragent` value and return the browser name and version. Lookups are covered later in this book.

There's more...

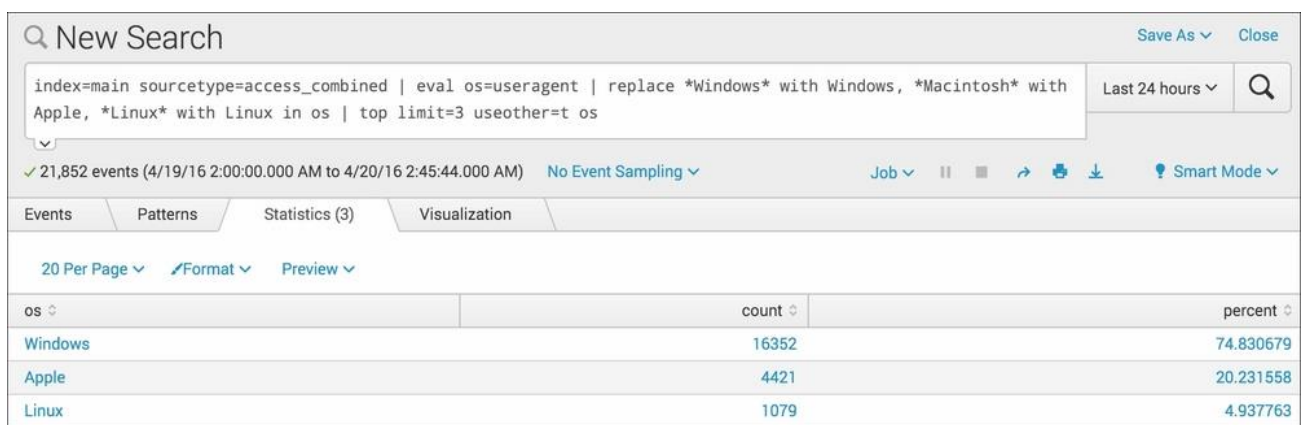
Often, the same field values can be used in different ways to provide additional insight. In this case, the `useragent` field can be used to inform the types of devices that access our site.

SEARCHING FOR THE WEB BROWSER DATA FOR THE MOST USED OS TYPES

Let's modify the search to display the types of user operating systems that access our website:

```
index=main sourcetype=access_combined | eval os=useragent |  
replace *Windows* with Windows, *Macintosh* with Apple,  
*Linux* with Linux in os | top limit=3 useother=t os
```

When the search is run, you should see results similar to the following screenshot:



New Search			
index=main sourcetype=access_combined eval os=useragent replace *Windows* with Windows, *Macintosh* with Apple, *Linux* with Linux in os top limit=3 useother=t os			Last 24 hours
21,852 events (4/19/16 2:00:00.000 AM to 4/20/16 2:45:44.000 AM) No Event Sampling			
Events	Patterns	Statistics (3)	Visualization
20 Per Page Format Preview			
os	count	percent	
Windows	16352	74.830679	
Apple	4421	20.231558	
Linux	1079	4.937763	

The search is similar, but this time we decided to pull the OS-related information from the `useragent` field and used it to compare access between major OS types.

Identifying the top-referring websites

Our web access logs continue to give us great information about our website and the users visiting the site. Understanding where our users are coming from provides insight into potential sales leads and/or which marketing activities might be working better over others. For this information, we look for the `referrer_domain` field value within the log data.

In this recipe, we will write a Splunk search to find the top-referring websites.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from *Session 3-1, Play Time – Getting Data In*. You should be familiar with the Splunk search bar and the time range picker.

How to do it...

Follow the given steps to search for the top-referring websites:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | stats dc(clientip) AS  
Referrals by referrer_domain | sort - Referrals
```

4. Splunk will return a tabulated list ordered by the number of unique referrals each website has provided.

New Search		Save As	Close
index=main sourcetype=access_combined stats dc(clientip) AS Referrals by referer_domain sort - Referrals		Last 24 hours	Q
✓ 21,876 events (4/19/16 2:00:00.000 AM to 4/20/16 2:47:20.000 AM) No Event Sampling		Job	Smart Mode
Events	Patterns	Statistics (11)	Visualization
20 Per Page		Format	Preview
referrer_domain	Referrals		
http://www.bing.com	350		
https://www4.samplesite.ca	362		
https://www2.samplesite.ca	365		
http://www.yahoo.com	367		
http://www.google.ca	383		
http://www.aol.com	394		
https://www3.samplesite.ca	395		

5. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_top_referring_websites` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main</code> <code>sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this session.
<code> stats dc(clientip) AS Referrals by referer_domain</code>	Using the <code>stats</code> command, we apply the distinct count (<code>dc</code>) function to <code>clientip</code> to count the unique IP addresses by <code>referrer_domain</code> and rename the generated count field to <code>Referrals</code> .
<code> sort - Referrals</code>	Using the <code>sort</code> command, we sort by the number of referrals in the descending order.

There's more...

In this recipe, we did not use the `top` command, as this command only provides limited functionality. The `stats` command is far more powerful and has many available functions, including distinct count.

SEARCHING FOR THE TOP 10 USING STATS INSTEAD OF TOP

Using the `stats` command in this recipe, we brought back all the websites present in our web access logs and then sorted them by the number of unique referrals. Should we want to only show the top 10, we can simply add the `head` command at the end of our search, as follows:

```
index=main sourcetype=access_combined | stats dc(clientip)
AS Referrals by referer_domain | sort - Referrals | head 10
```

The `head` command keeps the first specified number of rows. In this case, as we have a descending sort, by keeping the first 10 rows, we are essentially keeping the top 10. Instead of using the `head` command, we can also use the `limit` parameter of the `sort` command, as follows:

```
index=main sourcetype=access_combined | stats dc(clientip)
AS Referrals by referer_domain | sort - Referrals limit=10
```

NOTE

There is a great guide in the Splunk documentation to understand all the different functions for `stats`, `chart`, and `timechart`, available at <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/CommonStatsFunctions>.

Charting web page response codes

Log data often contains seemingly cryptic codes that have all sorts of meanings. This is true of our web access logs, where there is a status code that represents a web page response. This code is very useful as it can tell us whether certain events were successful or not. For example, error codes found in purchase events are less than ideal, and if our website was at fault, then we might have lost a sale.

In this recipe, we will write a Splunk search to chart web page responses against the various web pages on the site.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker.

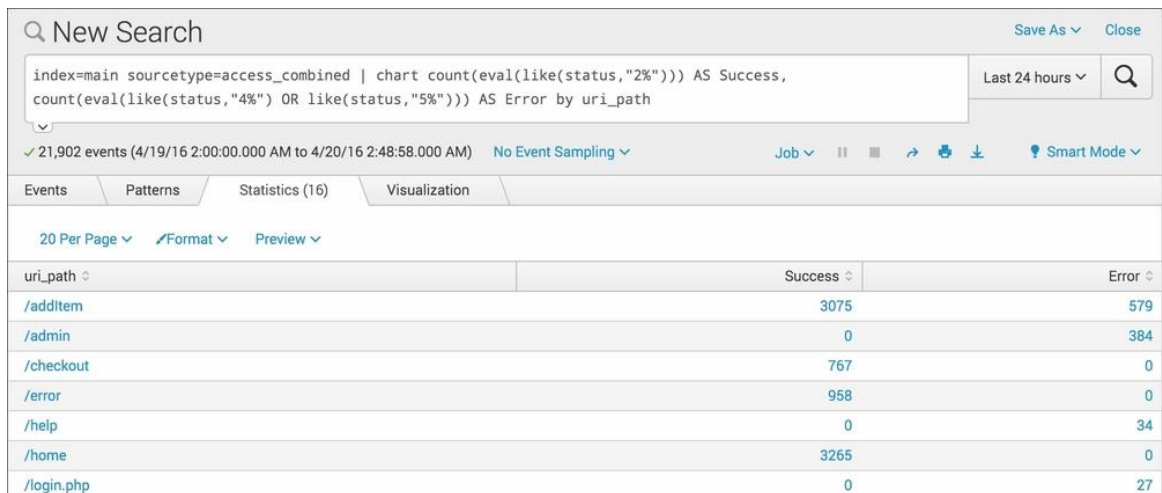
How to do it...

Follow the given steps to chart the web page response codes over time:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | chart  
count(eval(like(status,"2%"))) AS Success,  
count(eval(like(status,"4%") OR like(status,"5%"))) AS Error by  
uri_path
```

4. Splunk will return a tabulated list of web pages, detailing for each page how many events were successful and how many generated errors.



5. Click on the **Visualization** tab, and you will see this data represented in a column (by default) chart.
6. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_webpage_response_codes` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

In this recipe, we selected to search by the `uri_path` field. This field represents the various web pages on the site. Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this session.
<code> chart count(eval(like(status,"2%"))) AS Success, count(eval(like(status,"4%") OR like(status,"5%"))) AS Error by uri_path</code>	Stripping away the complexity for a moment, this is very similar to performing stats count by <code>uri_path</code> . However, in this case, we are using the <code>chart</code> command and only counting success and error status codes. As the status field is essentially just a code, we are evaluating whether the code represents success or error. We

Search fragment	Description
	do this using an inline <code>eval</code> command with the <code>like</code> function. The <code>like</code> function allows us to specify the start of the status field value and then wildcard it with a <code>%</code> sign. Any status code beginning with <code>2</code> represents success events, and any status code beginning with either <code>4</code> or <code>5</code> represents an error.

There's more...

Hopefully, you can start to see the power of the SPL, as we start to ramp up the complexity a bit. We can now take this search a little further to provide a bit more insight.

TOTALING SUCCESS AND ERROR WEB PAGE RESPONSE CODES

We can further amend the search to show only the `addItem` and `checkout` web pages events, which seems a little more relevant to sales intelligence. Additionally, using the `addcoltotals` command, we can add up the total success and error events:

```
index=main sourcetype=access_combined uri_path="/addItem" OR
uri_path="/checkout" | chart count(eval(like(status,"2%")))
AS Success, count(eval(like(status,"4%") OR
like(status,"5%"))) AS Error by uri_path | addcoltotals
label=Total labelfield=uri_path
```

When this updated search is run, you should see results similar to the following screenshot:

<div> <div>New Search</div> <div> <div> <div>index=main sourcetype=access_combined uri_path="/addItem" OR uri_path="/checkout" chart count(eval(like(status,"2%"))) AS Success, count(eval(like(status,"4%") OR like(status,"5%"))) AS Error by uri_path addcoltotals label=Total labelfield=uri_path</div> <div></div> </div> <div>Last 24 hours</div> <div></div> </div> </div>		
<div> <div>5,001 events (4/19/16 2:00:00.000 AM to 4/20/16 2:50:23.000 AM)</div> <div>No Event Sampling</div> <div>Job</div> <div></div> <div></div> <div></div> <div></div> <div>Smart Mode</div> </div>		
<div> <div>Events</div> <div>Patterns</div> <div>Statistics (3)</div> <div>Visualization</div> </div>		
<div> <div>20 Per Page</div> <div>Format</div> <div>Preview</div> </div>		
uri_path	Success	Error
/addItem	3079	579
/checkout	768	0
Total	3847	579

We use `labelfield=uri_path` and `label=Total` to tell Splunk to place a value of **Total** in the **uri_path** field column.

Displaying web page response time statistics

No one likes to wait for a web page to load, and we certainly do not want users of our web application waiting either! Within our web access logs, there is a field named `response` that tracks the total time the page has taken to load in milliseconds.

In this recipe, we will track the average page load time over the past week at different times of the day.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from **Session 3-1, *Play Time – Getting Data In***. You should be familiar with the Splunk search bar and the time range picker.

How to do it...

Follow the given steps to search and calculate the web page response time statistics over the past week:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 7 days** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
sourcetype=access_combined | timechart span=6h avg(response) AS  
avg_response | eval avg_response=round(avg_response/1000,2)
```

4. Splunk will return a tabulated list, detailing the average response time for every 6-hour period, going back a week.

New Search Save As Close

`sourcetype=access_combined | timechart span=6h avg(response) AS avg_response | eval avg_response=round(avg_response/1000,2)`
Last 7 days Q

✓ 149,052 events (4/13/16 2:00:00.000 AM to 4/20/16 2:52:05.000 AM) No Event Sampling
Job || ■ ↶ ↷ ⬇ 💡 Smart Mode

Events Patterns Statistics (29) Visualization

20 Per Page ✓ Format Preview
< Prev 1 2 Next >

_time	avg_response
2016-04-13 00:00	0.04
2016-04-13 06:00	0.05
2016-04-13 12:00	0.04
2016-04-13 18:00	0.04
2016-04-14 00:00	0.04
2016-04-14 06:00	0.04

- This is great, but hard to visualize in a tabular form. Click on the **Visualization** tab and you will see this data represented as a chart.
- Click on the chart type link in the upper-left of the chart (next to the **Format** link) and select **Line** if not already selected. Splunk now presents this data in a nice line chart, and we can now see the average response time at different times of the day much more clearly.



- Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_webpage_response_times` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this session.
<code> timechart span=6h avg(response) AS avg_response</code>	Using the <code>timechart</code> command, we specify a span of 6 hours. We then use the <code>avg</code> function on the response field. Splunk will add up all the response times in the 6-hour period and then calculate the average response time during that period.
<code> eval avg_response=round(avg_response/1000,2)</code>	Using the <code>eval</code> command, we calculate the average response time in seconds by dividing the average time (which is in milliseconds) by <code>1000</code> , to give us the time in seconds. The number <code>2</code> at the end is part of the round function and tells Splunk to round to <code>2</code> decimal places.

There's more...

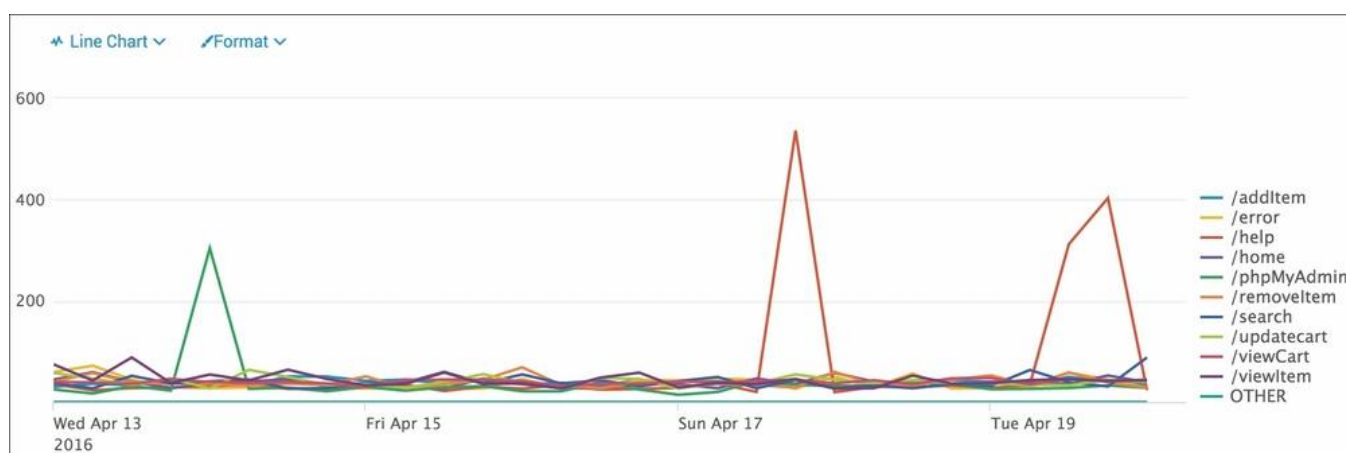
The `timechart` command offers some great functionality. Searches like this can be extended further to graphically compare several weeks against one another to spot anomalies and other issues.

DISPLAYING WEB PAGE RESPONSE TIME BY ACTION

We can further amend the search to offer granular information on average response time by the type of action being performed. This might pinpoint some actions that are less responsive than the others. For example, we might want to ensure that the checkout page remains at an optimal load time. For the following search to work, you must complete the *Defining field extractions* recipe in Session 3-1, *Play Time – Getting Data In*, to extract the response field:


```
sourcetype=access_combined uri_path=* | timechart span=6h
avg(response) by uri_path | foreach * [eval
<<FIELD>>=round(<<FIELD>>/1000,2) ]
```

We are now searching for web page events and then we will calculate the average time by page (`uri_field`). This results in a table of multiple columns, where each column represents a different web page. When we visualize this on a line graph, we now see many different lines on the same chart—pretty cool! You will notice that we used a pretty advanced Splunk search command, earlier named `foreach`. This is essentially a for type loop that cycles through each of the column fields in the table and applies a calculation to convert the average time by page from milliseconds to seconds while rounding the value to two decimal places:



Listing the top viewed products

Our web access logs capture the product IDs (the item field in the logs) that the users view and add to their shopping carts. Understanding the top products that people view can help influence our sales and marketing strategy, and even product direction. Products viewed on an e-commerce website might not always necessarily translate into sales of that product though.

In this recipe, we will write a Splunk search to chart the top 10 products that users successfully view and compare against the number of successful shopping cart additions for each product. For example, if a product has a high number of views but the product is not added to carts and subsequently purchased, this could indicate that something is not right—perhaps the pricing of the product is too high.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker.

How to do it...

Follow the given steps to search for the top products being searched over the past week:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 7 days** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined uri_path="/viewItem" OR  
uri_path="/addItem" status=200 | dedup JSESSIONID uri_path  
item | chart count(eval(uri_path="/viewItem")) AS view,  
count(eval(uri_path="/addItem")) AS add by item | sort - view |  
head 10
```

- Splunk will return a tabulated list of items (products), detailing the number of times a product was successfully viewed versus the number of times that product was actually added to the shopping cart.

The screenshot shows the Splunk Search interface. At the top, there's a search bar with the query: `index=main sourcetype=access_combined uri_path="/viewItem" OR uri_path="/addItem" status=200 | dedup JSESSIONID uri_path item | chart count(eval(uri_path="/viewItem")) AS view, count(eval(uri_path="/addItem")) AS add by item | sort - view | head 10`. Below the search bar, it shows 34,015 events. The results are displayed in a table with columns: item, view, and add.

item	view	add
38492	0	3117
1000016	0	2972
1000020	0	3070
1000014	4279	3846
1000015	4396	3957
4728475	4407	3971

- Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_top_products_viewed` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

In this recipe, our search returned a count by item of how many items were viewed versus how many were added to the cart. In this case, the item field represents a unique item ID that pertains to a specific product. Let's break down the search piece by piece:

Search fragment	Description
<code>index=main</code> <code>sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this session.
<code>uri_path="/viewItem" OR</code> <code>uri_path="/addItem" status=200</code>	Following best practice of making our search as granular as possible, we only search for events that contain <code>uri_paths</code> related to viewing items and adding items, and have a successful status code of <code>200</code> . This type of granularity greatly limits the number of

Search fragment	Description
	records we search, making our search a lot faster.
<code> dedup JSESSIONID uri_path item</code>	Using the <code>dedup</code> command, we de-duplicate our data by the <code>JSESSIONID</code> , the <code>uri_path</code> , and the <code>item</code> values. Why? Well, because a user in a given session could view a product many times in that session before adding it, so we want to ensure that we only count one view and one addition per user session of a product.
<code> chart count(eval(uri_path="/viewItem")) AS view, count(eval(uri_path="/addItem")) AS add by item</code>	Using the <code>chart</code> and <code>eval</code> commands, we count the number of views and adds by item.
<code> sort - view head 10</code>	Using the <code>sort</code> command, we sort in descending order on the <code>view</code> field, such that the items with the most number of views are at the top. We then leverage the <code>head</code> command to keep only the first 10 rows of data, leaving us with the top 10 searched products.

There's more...

This recipe provides us with some insight into product views and subsequent shopping cart additions that might then lead on to a sale. However, we can keep adding to the search to make it even easier to understand the relationship between the two.

SEARCHING FOR THE PERCENTAGE OF CART ADDITIONS FROM PRODUCT VIEWS

We can further amend the search from this recipe to evaluate a new column that calculates the percentage of product views added to the

cart. We do this using the `eval` command and some basic math, as follows:

```
index=main sourcetype=access_combined uri_path="/viewItem"  
OR uri_path="/addItem" status=200 | dedup JSESSIONID  
uri_path item | chart count(eval(uri_path="/viewItem")) AS  
view, count(eval(uri_path="/addItem")) AS add by item | sort  
- view | head 10 | eval  
cart_conversion=round(add/view*100)."%"
```

We firstly evaluate a new field called `cart_conversion` and then calculate the percentage, dividing purchase by view and multiplying by 100. We use the round function of `eval` to eliminate the decimal places and then tack on the `%` character at the end. Now, we can easily see what percentage of views lead to cart additions.

Charting the application's functional performance

Another of the data samples we loaded in Session 3-1, Play Time – Getting Data In, contained application logs from our application server. These have a Splunk sourcetype of `log4j` and detail the various calls that our application makes to the backend database in response to user web requests, in addition to providing insight into memory utilization and other health-related information. We are particularly interested in tracking how our application is performing in relation to the time taken to process user-driven requests for information.

In this recipe, we will write a Splunk search to find out how our application is performing. To do this, we will analyze database call transactions and chart the maximum, mean, and minimum transaction durations over the past week.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from Session 3-1, Play Time – Getting Data In. You should be familiar with the Splunk search bar and the time range picker.

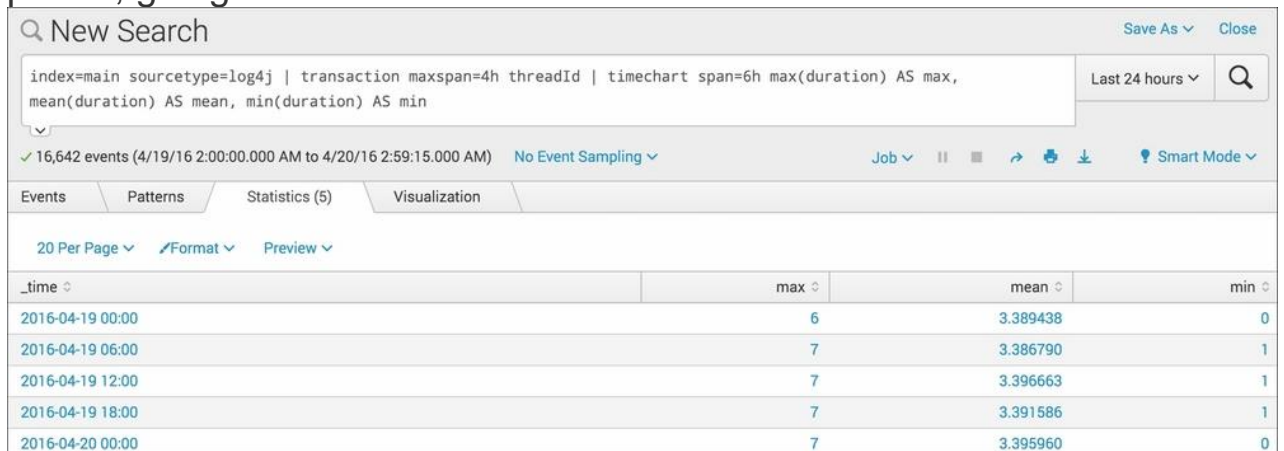
How to do it...

Follow the given steps to chart the application's functional performance over the past week:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=log4j | transaction maxspan=4h threadId |  
timechart span=6h max(duration) AS max, mean(duration) AS mean,  
min(duration) AS min
```

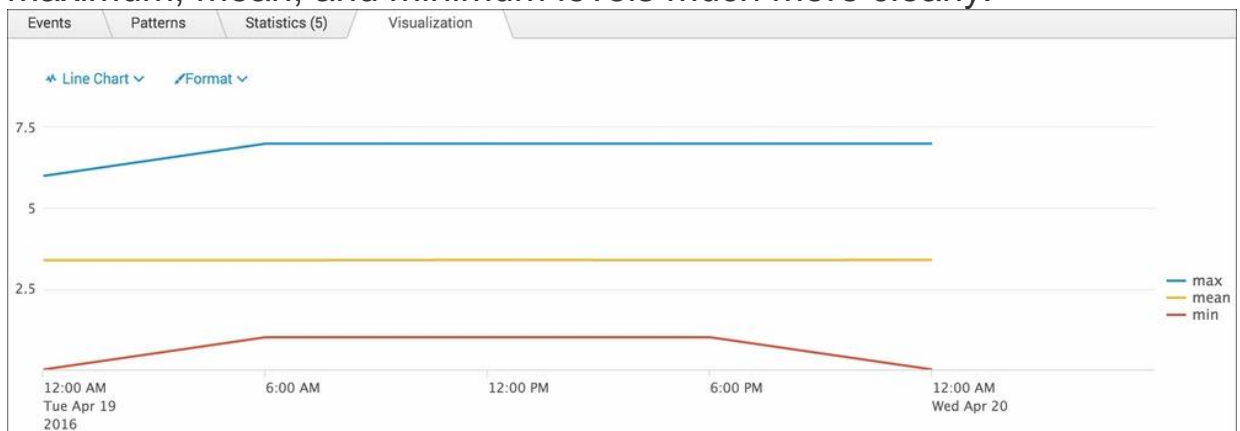
- Splunk will return a tabulated list, detailing the maximum, mean, and minimum database transaction durations for every 6-hour period, going back the last 24 hours.



The screenshot shows the Splunk search interface. The search bar contains the query: `index=main sourcetype=log4j | transaction maxspan=4h threadId | timechart span=6h max(duration) AS max, mean(duration) AS mean, min(duration) AS min`. The results show 16,642 events from 4/19/16 2:00:00.000 AM to 4/20/16 2:59:15.000 AM. The table below displays the data for the last 24 hours.

_time	max	mean	min
2016-04-19 00:00	6	3.389438	0
2016-04-19 06:00	7	3.386790	1
2016-04-19 12:00	7	3.396663	1
2016-04-19 18:00	7	3.391586	1
2016-04-20 00:00	7	3.395960	0

- This is great, but hard to visualize in a tabular form. Click on the **Visualization** tab and you will see this data represented as a chart.
- Click on the chart type link in the upper-left of the chart (next to the **Format** link) and select **Line** if not already selected. Splunk now presents this data in a nice line chart, and we can now see the maximum, mean, and minimum levels much more clearly.



- Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_application_performance` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
index=main sourcetype=log4j	In this example, we search for our application logs which have the <code>log4j</code> sourcetype.
transaction maxspan=4h threadId	Using the <code>transaction</code> command, we essentially consolidate multiple events with a common <code>threadId</code> into single event, multiline transactions. The <code>maxspan</code> function tells Splunk to only look at events with the same <code>threadId</code> that are within 4 hours of each other. The <code>transaction</code> command also calculates a new field called <code>duration</code> . This is the <code>duration</code> in seconds from the first event in the transaction to the last event in the transaction.
timechart span=6h max(duration) AS max, mean(duration) AS mean, min(duration) AS min	Using the <code>timechart</code> command, we specify a span of 6 hours. We then use the <code>max</code> , <code>mean</code> , and <code>min</code> functions on the <code>duration</code> field. Splunk analyzes the durations in the 6-hour period and then calculates the max, mean, and min durations during this period.

TIP

The `transaction` command is an extremely resource intensive (CPU/memory) search command. When using this command, be sure to use the `maxspan` function where possible, as this helps focus on transactions grouped only within the specified `maxspan` timeframe.

There's more...

In this recipe, we leveraged the `transaction` command. This is a very useful and powerful function, and we will revisit it in more depth and complexity later in the book.

Charting the application's memory usage

In addition to measuring functional performance of database transactions, we are also interested in understanding how our application is performing from a memory usage perspective. Analyzing this type of information can help identify memory leaks in our application or high-memory utilization that might be affecting the user experience and causing our application to slow down.

In this recipe, we will analyze the memory usage of our application over time.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker.

How to do it...

Follow the given steps to chart the application memory usage over the past day:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=log4j perfType="MEMORY" | eval  
mem_used_pc=round((mem used/mem total)*100) | eval  
mem_remain_pc=(100-mem_used_pc) | timechart span=15m  
avg(mem_remain_pc) avg(mem_used_pc)
```

4. Splunk will return a tabulated list, detailing all the events that meet our search criteria.

New Search Save As Close

index=main sourcetype=log4j perfType="MEMORY" | eval mem_used_pc=round((mem_used/mem_total)*100) | eval mem_remain_pc=(100-mem_used_pc) | timechart span=15m avg(mem_remain_pc) avg(mem_used_pc)

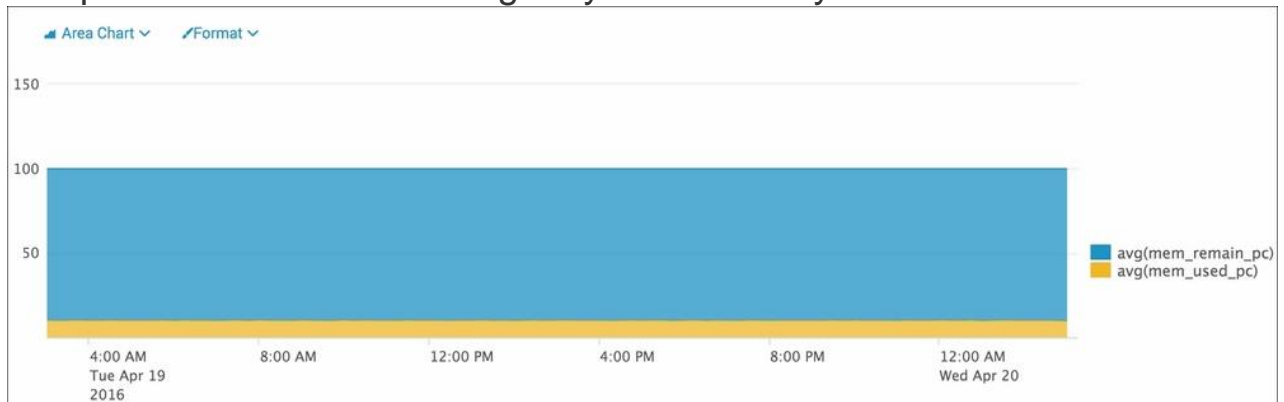
16,053 events (4/19/16 3:00:00.000 AM to 4/20/16 3:02:05.000 AM) No Event Sampling Job || ↶ ↷ ⬇ Smart Mode

Events Patterns Statistics (97) Visualization

20 Per Page Format Preview < Prev 1 2 3 4 5 Next >

_time	avg(mem_remain_pc)	avg(mem_used_pc)
2016-04-19 03:00:00	89.880952	10.119048
2016-04-19 03:15:00	90.113772	9.886228
2016-04-19 03:30:00	89.994083	10.005917
2016-04-19 03:45:00	89.880952	10.119048
2016-04-19 04:00:00	90.048193	9.951807

- This is great, but hard to visualize in a tabular form. Click on the **Visualization** tab and you will see this data represented in a column (by default) chart.
- Click on the column link the chart and select **Area**. Then, click on the **Format** link and change **Stack Mode** to **Stacked** and click on **Apply**. Splunk now presents this data in an area chart, allowing us to easily see if there are times during the day when our application might be getting low on memory. In this case, our sample data looks to be using very little memory.



- Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_application_memory` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=log4j perfType="MEMORY"</code>	In this example, we search for our application logs which have the <code>log4j</code> sourcetype. We also select to view only the memory-related events.
<code> eval mem_used_pc=round((mem_used/mem_total)*100)</code>	Using the <code>eval</code> command, we calculate the percentage of memory used from the <code>mem_used</code> and <code>mem_total</code> fields in our application log.
<code> eval mem_remain_pc=(100-mem_used_pc)</code>	Using the <code>eval</code> command again, we calculate the remaining percentage of memory from the used percentage of memory that we just calculated in the previous step.
<code> timechart span=15m avg(mem_remain_pc) avg(mem_used_pc)</code>	Using the <code>timechart</code> command, we calculate the average remaining percentage of memory and the used percentage of memory for every 15-minute interval over the past day.

Counting the total number of database connections

Our application only allows for a limited number of concurrent database connections currently. As our application user base grows, we need to proactively monitor these connections to ensure that we do not hit our concurrency limit or to know when we need to further scale out the database infrastructure.

In the last recipe of this session, we will monitor database transactions over the past week to identify if there are certain times or days when we might be close to our concurrency limit.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from **Session 3-1, *Play Time – Getting Data In***. You should be familiar with the Splunk search bar and the time range picker.

How to do it...

Follow the given steps to search for the total number of database connections over the past 30 days:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 7 days** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=log4j perfType="DB" | eval  
threshold=con_total/100*70 | where con_used>=threshold |  
timechart span=4h count(con_used) AS CountOverThreshold
```

4. Splunk will return a tabulated list, detailing all the events that meet our search criteria.

New Search Save As Close

index=main sourcetype=log4j perfType="DB" | eval threshold=con_total/100*70 | where con_used>=threshold | timechart span=4h count(con_used) AS CountOverThreshold

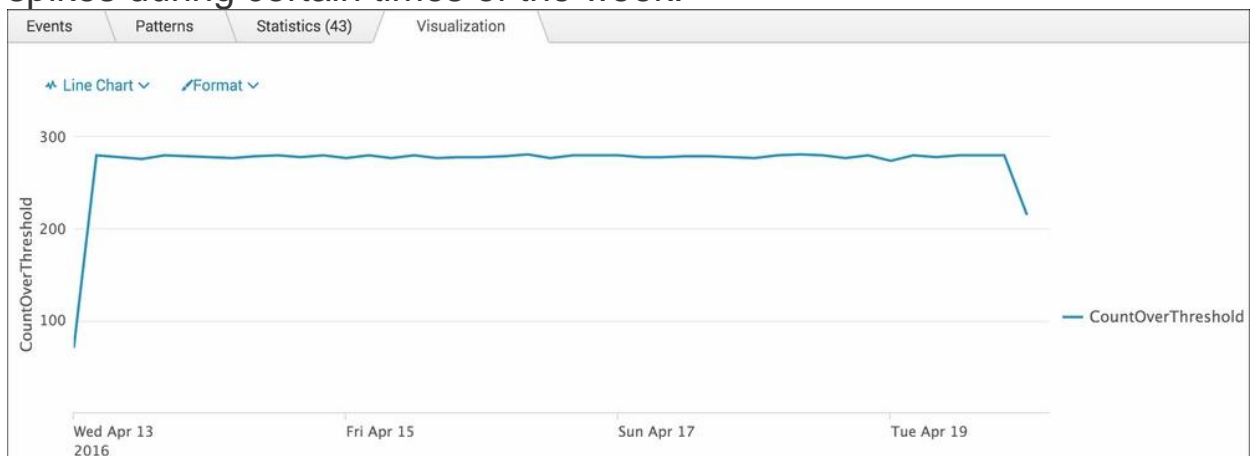
11,713 events (4/13/16 3:00:00.000 AM to 4/20/16 3:05:05.000 AM) No Event Sampling Job || → ⬇ Smart Mode

Events Patterns Statistics (43) Visualization

20 Per Page Format Preview < Prev 1 2 3 Next >

_time	CountOverThreshold
2016-04-13 00:00	70
2016-04-13 04:00	280
2016-04-13 08:00	278
2016-04-13 12:00	276
2016-04-13 16:00	280
2016-04-13 20:00	279
2016-04-14 00:00	278

- This is great, but hard to visualize in a tabular form. Click on the **Visualization** tab and you will see this data represented in a column (by default) chart.
- Click on the column link in the chart and select **Line**. Splunk now presents this data in a line chart, allowing us to easily see any spikes during certain times of the week.



- Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_application_db_connections` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=log4j perfType="DB"</code>	In this example, we search for our application logs which have the <code>log4j</code> sourcetype. We also select to view only the events related to database (<code>DB</code>).
<code> eval threshold=con_total/100*70</code>	Using the <code>eval</code> command, we calculate a new field called <code>threshold</code> , which is 70 percent of the total connections permitted.
<code> where con_used>=threshold</code>	Using the <code>where</code> command, we search for only those events that are greater than or equal to the 70 percent threshold we just defined.
<code> timechart span=4h count(con_used) AS CountOverThreshold</code>	Finally, we count the number of times over a 4-hour period in which the connection limit is greater than or equal to our threshold.