Master Joining Datasets Without Using Join

Nick Mealy

CEO, Chief Mad Scientist | Sideview, LLC

splunk> TRUST

splunk> .conf19

© 2019 SPLUNK INC.

Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements we may make. In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

Splunk, Splunk>, Listen to Your Data, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2019 Splunk Inc. All rights reserved.

© 2019 SPLUNK INC.

Why is this guy qualified to give this talk?

Former Splunk Mad Scientist and Principal UI Developer 2005-2010

This is Rob and I in the booth at Linuxworld in 2005, the week we launched Splunk 1.0.

© 2019 SPLUNK INC.

Why is this guy qualified to give this talk?

Whenever there was any new search and reporting functionality in Splunk, the UI team was the first thrown into the pit.

The first people to hit the bottom learned how to welcome everyone else.

© 2019 SPLUNK INC.

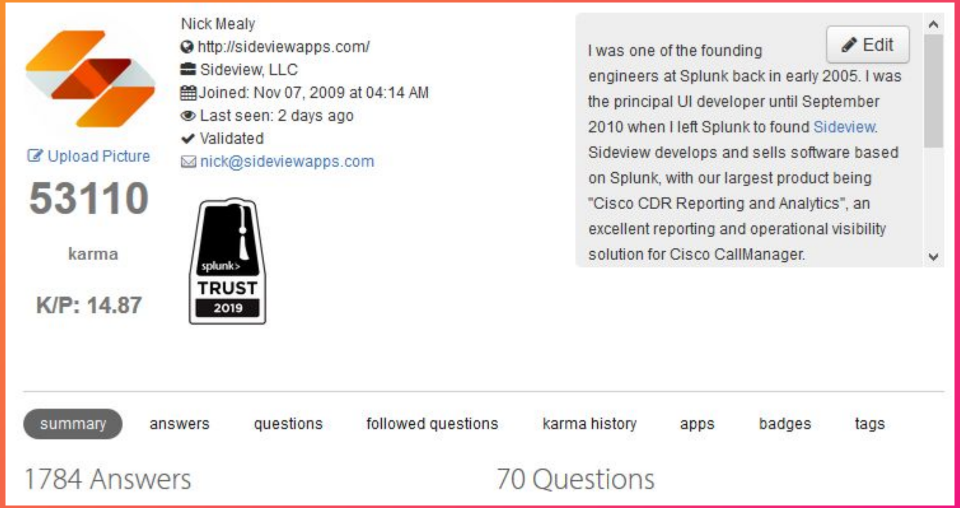
Why is this guy qualified to give this talk?

This is me gearing up when the whole company went spelunking together.

Only two people vomited that I know of.

© 2019 SPLUNK INC.

OK so… I heard 2010. How about the last 9 years?



Nick Mealy
http://sideviewapps.com/
Sideview, LLC
Joined: Nov 07, 2009 at 04:14 AM
Last seen: 2 days ago
Validated
nick@sideviewapps.com

Upload Picture
53110
karma
K/P: 14.87

TRUST 2019

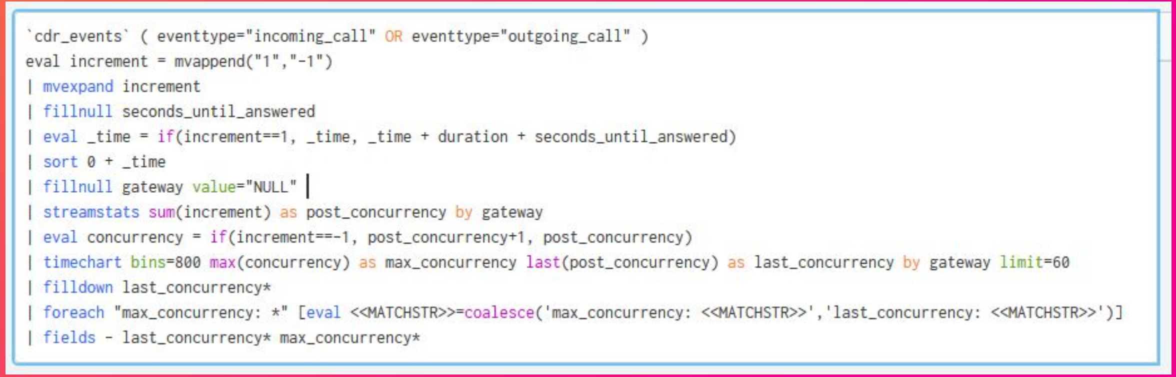
Edit

I was one of the founding engineers at Splunk back in early 2005. I was the principal UI developer until September 2010 when I left Splunk to found Sideview. Sideview develops and sells software based on Splunk, with our largest product being "Cisco CDR Reporting and Analytics", an excellent reporting and operational visibility solution for Cisco CallManager.

summary answers questions followed questions karma history apps badges tags

1784 Answers 70 Questions

For many years I was one of the folks who tended to answer the complex SPL and postprocess questions on answers.splunk.com.



```
`cdr_events` ( eventtype="incoming_call" OR eventtype="outgoing_call" )
eval increment = mvappend("1","-1")
| mvexpand increment
| fillnull seconds_until_answered
| eval _time = if(increment==1, _time, _time + duration + seconds_until_answered)
| sort 0 + _time
| fillnull gateway value="NULL"
| streamstats sum(increment) as post_concurrency by gateway
| eval concurrency = if(increment==-1, post_concurrency+1, post_concurrency)
| timechart bins=800 max(concurrency) as max_concurrency last(post_concurrency) as last_concurrency by gateway limit=60
| filldown last_concurrency*
| foreach "max_concurrency: *" [eval <<MATCHSTR>>=coalesce('max_concurrency: <<MATCHSTR>>','last_concurrency: <<MATCHSTR>>')]
| fields - last_concurrency* max_concurrency*
```

Our main product for Cisco CallManager has to do some really hairy SPL

© 2019 SPLUNK INC.

How about us – why are we here?

Things you might come away from this talk with.

A) **Why join and append are evil.**
(as a bonus, why transaction is chaotic neutral)

B) How to see how much of the actual work your searches are pushing out to the indexers. [redacted]

C) A tendency to say "I wonder if we can use some conditional eval to fix this."

© 2019 SPLUNK INC.

First – some vocab

This talk is a bit advanced, but its also about things everyone would be better off knowing early.

We're going to have to use some fancy words in this talk. If you don't know what they mean right now, that's OK.
Get a copy of the slides and after the fake ending slide there will be links to splunk docs and things like that.

- MapReduce, subsearch, finalizing, autofinalizing
- The join, append, stats and transaction commands.
- Splunks official docs about "grouping" data from different sourcetypes.

© 2019 SPLUNK INC.

Part 1 – Why the bad things are bad.

splunk> .conf19

© 2019 SPLUNK INC.

Let's take a quick poll

How many of you –

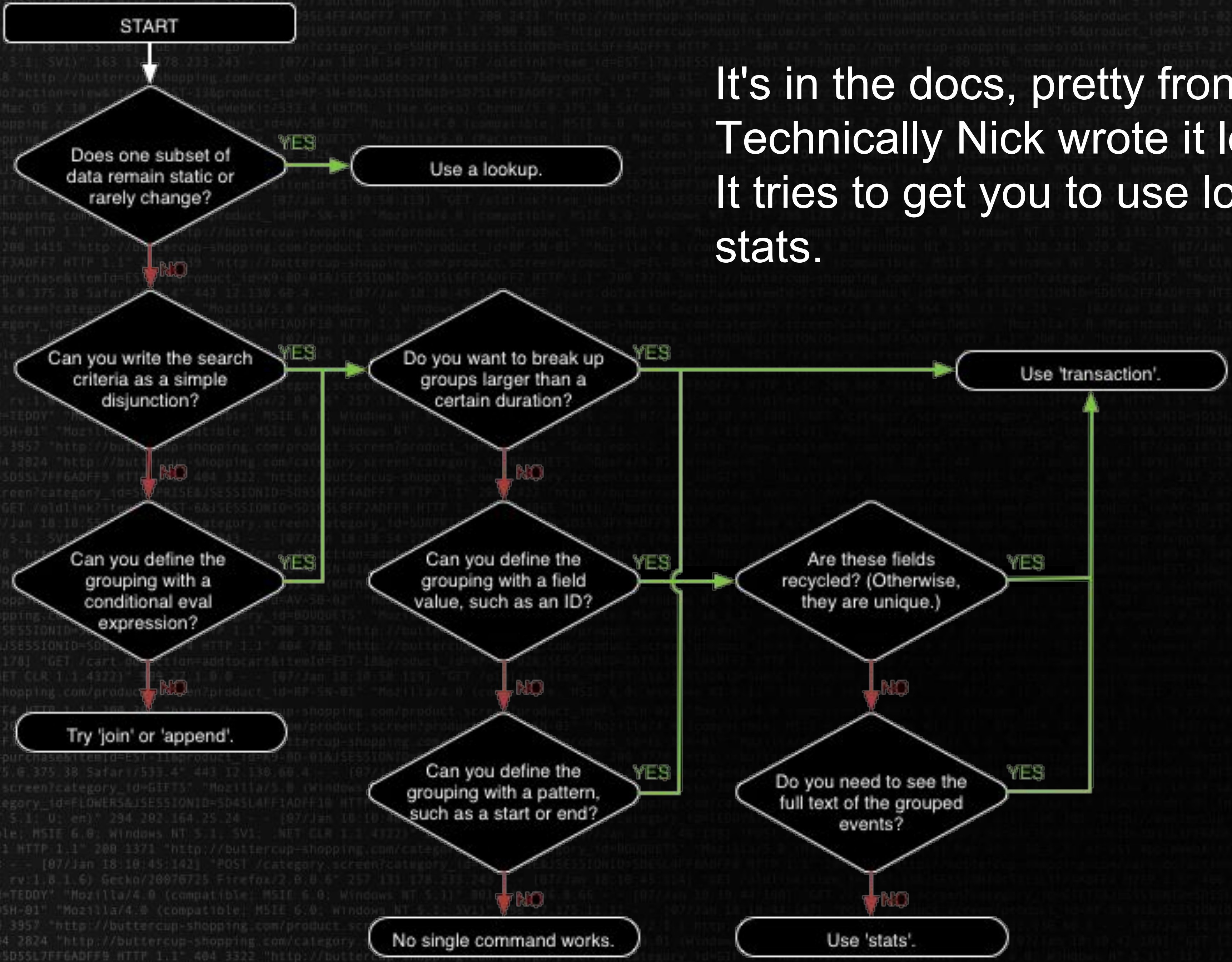
1. **personally run fairly expensive searches on your Splunk instances that use join, append, appendcols or transaction?**

2. **administer Splunk instances where searches like that are running?**

3. **have the feeling these searches might be 2x or 10x faster if rewritten more cleverly?**

4. **have reports where you'd love to run them over longer timeranges or involve more sourcetypes if they would just run faster or not hit truncation errors?**

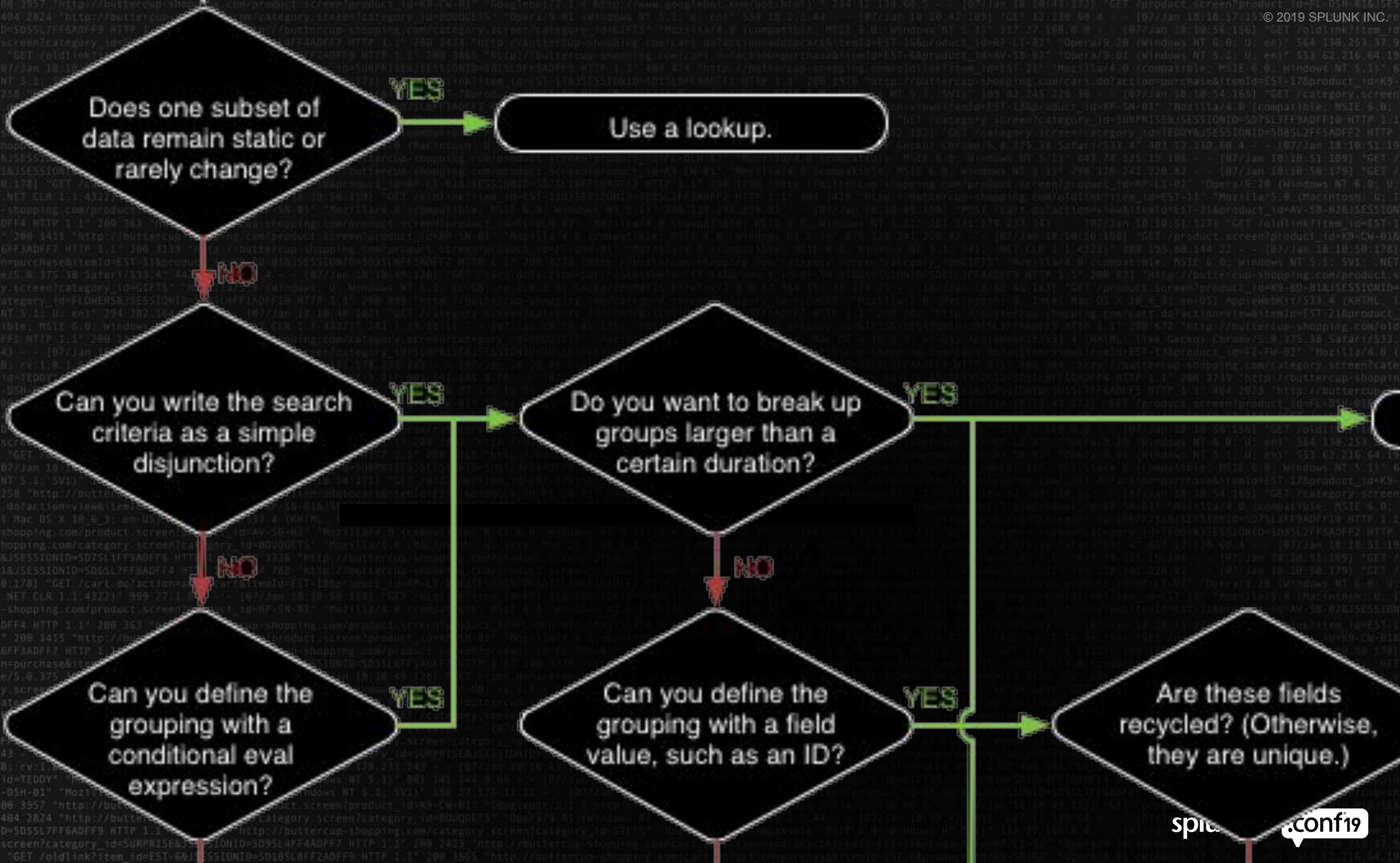
© 2019 SPLUNK INC.


START
Does one subset of data remain static or rarely change?
YES
Use a lookup.
NO
Can you write the search criteria as a simple disjunction?
YES
Do you want to break up groups larger than a certain duration?
YES
Use 'transaction'.
NO
Can you define the grouping with a conditional eval expression?
YES
NO
Try 'join' or 'append'.
NO
Can you define the grouping with a field value, such as an ID?
YES
Are these fields recycled? (Otherwise, they are unique.)
YES
NO
Can you define the grouping with a pattern, such as a start or end?
YES
NO
No single command works.
NO
Do you need to see the full text of the grouped events?
YES
NO
Use 'stats'.

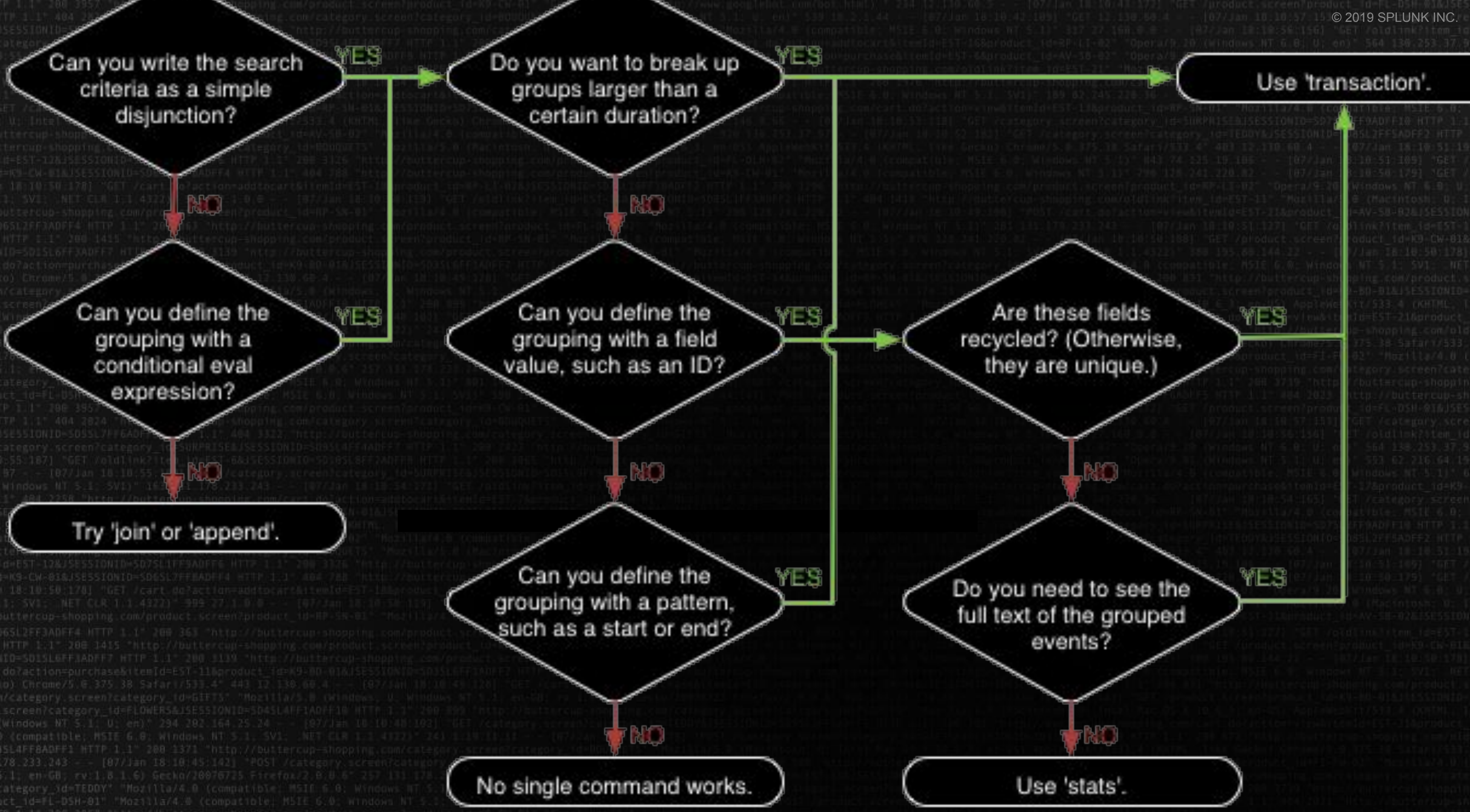
It's in the docs, pretty front and center.
Technically Nick wrote it long ago.
It tries to get you to use lookups and stats.

splunk> .conf19

© 2019 SPLUNK INC.


Does one subset of data remain static or rarely change?
YES
Use a lookup.
NO
Can you write the search criteria as a simple disjunction?
YES
Do you want to break up groups larger than a certain duration?
YES
NO
NO
Can you define the grouping with a conditional eval expression?
YES
Can you define the grouping with a field value, such as an ID?
YES
Are these fields recycled? (Otherwise, they are unique.)

© 2019 SPLUNK INC.


Can you write the search criteria as a simple disjunction?
YES
NO
Can you define the grouping with a conditional eval expression?
YES
NO
Try 'join' or 'append'.
Do you want to break up groups larger than a certain duration?
YES
NO
Can you define the grouping with a field value, such as an ID?
YES
NO
Can you define the grouping with a pattern, such as a start or end?
YES
NO
No single command works.
Use 'transaction'.
Are these fields recycled? (Otherwise, they are unique.)
YES
NO
Do you need to see the full text of the grouped events?
YES
NO
Use 'stats'.

© 2019 SPLUNK INC.

Naming things is hard

When you're starting out the names themselves can send users the wrong way.

What would SQL do? -- you will search the splunk docs for “join”.

docs are using this word “transaction”. -- "got it. there's a transaction command".

I need to like… tack on another column. – woo hoo "appendcols" ftw!!

vs

Stats? -- "nah, I don't need statistics right now, I need to group things."

NO. Stats eval and lookups should be your first tools.

Append/appendcols/join and even transaction should be last resorts.

© 2019 SPLUNK INC.

What's wrong with the join and append commands?

Fundamentally slow, and as soon as you push any real volume of data through them, they quietly break.

- results are truncated if you exceed 50,000 rows.
- The search in square brackets is quietly "autofinalized" when its execution time exceeds 120 seconds (or 60 seconds).
- 2 jobs instead of 1 means extra overhead.
- **You might not even *realize* that you're hitting autofinalize and row truncation limits, but your results are wrong.**
- **Breaking MapReduce. Forcing splunk to pull a lot more data back to the SH and do all the math on the SH.**
- **As a kind of "worst-practice", it proliferates quickly.**

© 2019 SPLUNK INC.

What's wrong with the transaction command?

- It's designed for edge cases - keeping all the arguments straight can be hard.
- It breaks MapReduce

If you're ever using transaction by some id field, and NOT also using any of the startswith / endswith / maxspan / maxpause args, then you can probably switch to stats or other core SPL that will work with MapReduce.

Here you really can go from "it takes 8 hours but at least it runs and it's right"
To
"That can't be right - it completes in 20 minutes now".

splunk> .conf19

© 2019 SPLUNK INC.

MapReduce – How Splunk's implementation works

```
 sourcetype=cdr type=outgoing | stats sum(duration) by
device_type
```

Say we want to see the sum of all call durations for each of 5 device_types, across a million calls stored in 10 indexers.

Let's imagine that WE are the search head. How do we do it?

WHAT'S THE WORST WAY POSSIBLE.

Let's ask the indexers to send us every single event…

AND we'll store this somewhere.

AND we'll do the search filtering ourselves.

AND then add up all the durations ourselves.

© 2019 SPLUNK INC.

MapReduce – How Splunk's implementation works

OK that was awful. We saturated our network, we didn't even have anywhere to store the data, and we had to do a ton of boring math on it.

```
sourcetype=cdr type=outgoing | stats sum(duration) by device_type
```

So let's at least send this part out, so the indexers can only send these events back to us

[redacted]

OK thank god. We are now at least doing "distributed search".

It's better but it still sucks. Something's missing.

We're still getting an ungodly number of rows, and doing a lot of math.

© 2019 SPLUNK INC.

MapReduce – How Splunk's implementation works

What if we could not only send our search terms to the indexers, but also somehow tell them to each give back Only a tiny summary table like this:

origDeviceName	sum(duration)
softphone	2422312
hardphone	858224
conference_bridge	582023
ip_communicator	590564
Jabber	18948

Then we'd only have to add up the totals from each of the 10 tiny tables?

THAT WOULD BE AWESOME. And this is what happens.

This is the **"distributed reporting"** part of Splunk, aka the "Reduce" part of its "MapReduce".

These little tables are sometimes called the "sufficient statistics" - half-baked cakes cooked by the indexers.

© 2019 SPLUNK INC.

MapReduce – How Splunk's implementation works
Let's review

"pre commands" = how the indexers know to send back only "sufficient statistics".

```
sourcetype=cdr type=outgoing      |      stats sum(duration) by device_type
```

distributable streaming portion

Will include all distributable streaming commands (eval, where, rename etc..)
Indexers run this part PLUS prestats

```
sourcetype=cdr type=outgoing
| prestats sum(duration) by
origDeviceName
```

Transforming portion

Starts at the first non-"distributable streaming" command, goes all the way to the end.

SH runs these commands at the end to tie it all together.

splunk> .conf19

© 2019 SPLUNK INC.

Which commands are "distributable streaming"?

Distributable Streaming ☺

```
Eval
where
search
rename
fillnull
fields
mvexpand
rex
…
```

TRANSFORMING Commands

with "pre" version ☺

```
Stats
chart
timechart
…
```

with no "pre" version ☹

```
join
append
transaction
table
eventstats
streamstats
…
```

https://docs.splunk.com/Documentation/Splunk/latest/Search/Typesofcommands

© 2019 SPLUNK INC.

And all this has been happening automatically!

Unless… you've inadvertently been writing suboptimal SPL.

Like that ever happens.

splunk> .conf19

© 2019 SPLUNK INC.

Part 2 – How to test whether something is breaking MapReduce

© 2019 SPLUNK INC.

First lets look at a "good" search

how to manually walk left to right in SPL to find first non-streaming command.

Open Job Inspector. Verify the 'remoteSearch' and 'reportSearch'.

Let's find the 'pre' command in the 'remoteSearch'

Now let's do a "bad" version with append.

let's get it to "autofinalize". If this was happening when it was scheduled you would never know.

Demo

© 2019 SPLUNK INC.

MapReduce – How to find out how you're doing

TEST IT!   The Job Inspector is your friend.

Click "Job" then "Inspect Job".

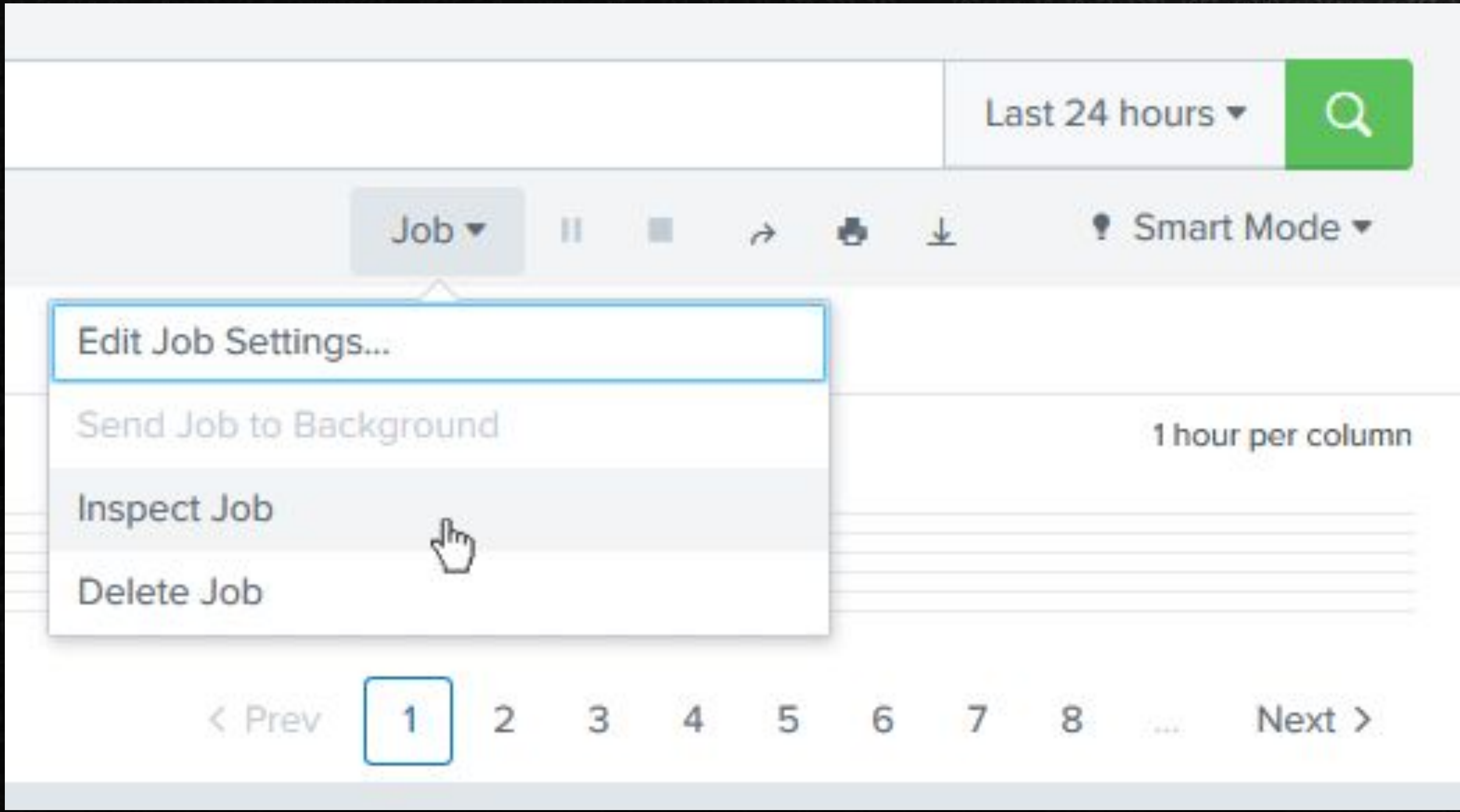
Scroll to the bottom

Click 'search job properties' to open the full set of keys.

Ctrl-F search for:

"remoteSearch" is what goes to the indexers.

"reportSearch" stays on the SH



splunk> .conf19

© 2019 SPLUNK INC.

MapReduce – How to find out how you're doing

Scanning from left to right, find the first command that is not "distributable streaming"

If that command has a "pre" command --   nice job!

If it doesn't, ie if it's join, append, transaction, table -- that's bad

Eg: all failed calls, inbound or outbound. Group by device and split by failure type.

```
sourcetype=cucm_cdr call_answerable=0 (type=outgoing OR type=incoming)
| eval device=if(type="outgoing",origDeviceName,destDeviceName)
| rename cause_description as failure
| chart dc(callId) over device by failure        << It's chart. Phew.
| addtotals
| sort - Total
```

splunk> .conf19

© 2019 SPLUNK INC.

MapReduce – How to find out how you're doing

TEST IT! The Job Inspector is your friend.

"remoteSearch" = what gets sent to the indexers. "reportSearch" = the part that runs on the SH.

Short version = look for a pre* command in remoteSearch.

With 85,000 events and ONLY ONE INDEXER:

```
  ( sourcetype=cucm_cdr OR sourcetype=cucm_cmr)
  | stats values(MLQK) as MLQK values(type) as type by globalCallID_callId
[redacted]
  | stats perc5(MLQK) by type
```

takes only 1.653 seconds

```
 sourcetype=cucm_cdr
  | join callId [search index=cisco_cdr host=cake sourcetype=cucm_cmr]
  | stats perc5(MLQK) by type
```

takes 15.026 seconds

© 2019 SPLUNK INC.

MapReduce – How to find out how you're doing

You can use the table command for good as well as for evil !

By using table to cripple MapReduce completely, you can measure how much work it was doing in the first place.

```
sourcetype=cucm_cdr | stats count by type
```

3.0 seconds

```
sourcetype=cucm_cdr | table type | stats count by type
```

17.6 seconds!

© 2019 SPLUNK INC.

MapReduce – How to find out how you're doing

On 5 million events and ONLY ONE INDEXER

```
sourcetype=cucm_cdr | stats count by type
```

61.8 seconds

```
sourcetype=cucm_cdr | table type | stats count by type
```

514 seconds !!!!!!!!

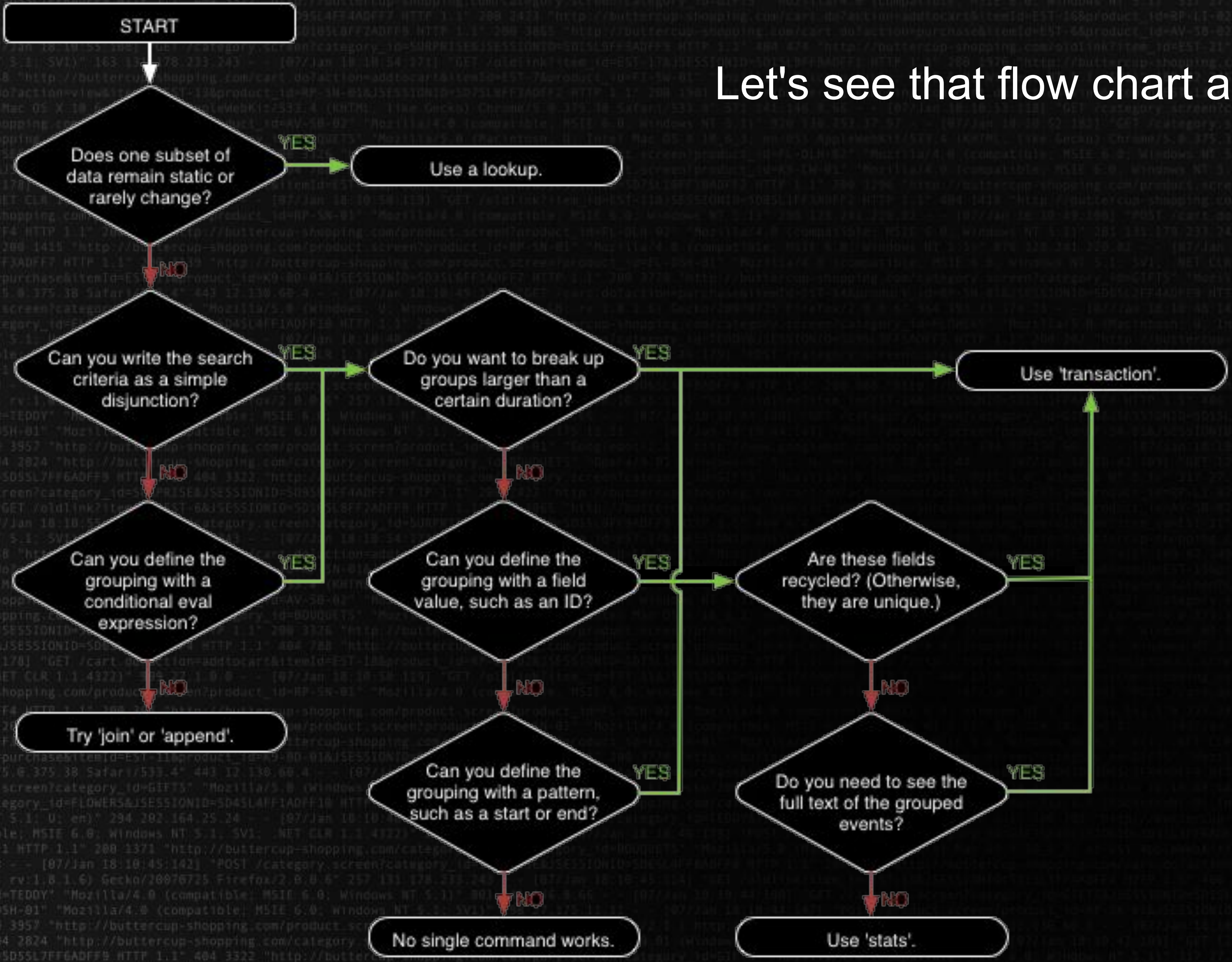
Effects are much more pronounced with more indexers.

Again these numbers are with ONE INDEXER.

splunk> .conf19

© 2019 SPLUNK INC.

Let's see that flow chart again.


START
Does one subset of data remain static or rarely change?
YES
Use a lookup.
NO
Can you write the search criteria as a simple disjunction?
YES
NO
Do you want to break up groups larger than a certain duration?
YES
Use 'transaction'.
NO
Can you define the grouping with a conditional eval expression?
YES
NO
Try 'join' or 'append'.
Can you define the grouping with a field value, such as an ID?
YES
NO
Are these fields recycled? (Otherwise, they are unique.)
YES
NO
Can you define the grouping with a pattern, such as a start or end?
YES
NO
No single command works.
Do you need to see the full text of the grouped events?
YES
NO
Use 'stats'.

© 2019 SPLUNK INC.

Sure. "use stats". But it's never that simple!!

- **Real data is a lot messier than this example.**

  Totally true.

- **The data cleanup/normalization/surgery that I need is easier with append/Join.**

  Also true – being able to use entirely different SPL on different sides is very nice.

- **They run fine on my dev server.**

  That's nice. At smaller scales appearances can deceive.

- **The "use stats" way seems correspondingly blocked because $reasons.**

  Yep. A lot of the "right" ways are pretty unintuitive. We'll get to some of them.

© 2019 SPLUNK INC.

Part 2 Conclusion – surface roads are a last resort

© 2019 SPLUNK INC.

Part 3 –'use stats'

Sure. But how? Let's look at some simple examples.

© 2019 SPLUNK INC.

Example #1

(I can't use stats) …cause I don't want to and hey there's a join command

Bad

```
sourcetype=db
| join pid [search sourcetype=app]
| stats sum(rows) sum(cputime) by pid
```

A little better

```
sourcetype=db | stats sum(rows) as rows by pid
| join pid [
    search sourcetype=app
    | stats sum(cputime) as cputime by pid
]
| stats sum(rows) sum(cputime) by pid
```

splunk> .conf19

© 2019 SPLUNK INC.

Example #1

(I can't use stats)   …cause I don't want to and hey there's a join command

Bad

```
sourcetype=db
| join pid [search sourcetype=app]
| stats sum(rows) sum(cputime) by pid
```

BEST

```
sourcetype=db OR sourcetype=app
| stats sum(rows) sum(cputime) by pid
```

[redacted]

© 2019 SPLUNK INC.

Example #2

… because I need to join first and THEN I need stats to do this other thing

AWFUL: 15.03s

```
sourcetype=cucm_cdr
| join callId [search sourcetype=cucm_cmr]
| stats perc5(MLQK) by type
```

Nope – just use stats twice.

Good 1.76s

```
sourcetype=cucm_cdr OR sourcetype=cucm_cmr
| stats values(MLQK) as MLQK
        last(type) as type
        by callId
| stats perc5(MLQK) by type
```

NOTE: values() is underrated for single valued fields – just because your field is single valued today doesn't mean it will be tomorrow.

© 2019 SPLUNK INC.

Example #3

…because the field names are annoyingly different

Let's say one side calls it “pid” and the other calls it “processId”

```
sourcetype=db
| rename processId as pid
| join pid [search sourcetype=app]
| stats sum(rows) sum(cputime) by pid
```

Just needs some "conditional eval".
Should be:

```
sourcetype=db OR sourcetype=app
| eval pid=if(sourcetype=="db",processId,pid)
| stats sum(rows) sum(cputime) by pid
```

Why if() and not coalesce()?

This amounts to a preference, but coalesce can betray you unexpectedly if assumptions change, or when the same coalesce statement is pasted around. Case() and if() are explicit and any assumptions they make are clearer to future readers.

splunk> .conf19

© 2019 SPLUNK INC.

Example #4

… because when I tried, it damaged some of the values I need.

I need some extra SPL on one side to clean up the data, but if it touches the other side it damages it.

```
sourcetype=db
| rex field=pid mode=sed "s/cruft//g"
| join pid [search sourcetype=app]
| stats sum(rows) sum(cputime) by pid
```

Just needs more conditional eval [redacted]

```
sourcetype=db OR sourcetype=app
| eval pid=if(sourcetype=="db",replace(pid,"cruft",""),pid)
| stats sum(rows) sum(cputime) by pid
```

ALSO you can sometimes use this to hide field(s) from the bad thing.

© 2019 SPLUNK INC.

Example #5

… because my data is a mess and I need to do surgery with join/append

I want to calculate things from different places but I need to do it cleanly. Join allows me to avoid random contamination.

Let's say sometimes sourcetype B events might have a "kb" field.

```
sourcetype=A | stats sum(kb) by ip
+
sourcetype=B | stats dc(sessionid) by ip
```

splunk> .conf19

© 2019 SPLUNK INC.

Example #5

… because my data is a mess and I need to do surgery with join/append

Solution: just more conditional eval. Kill the bad things.

```
sourcetype=A | stats sum(kb) by ip
+
sourcetype=B | stats dc(sessionid) by ip
=
sourcetype=A OR sourcetype=B
| eval kb=if(sourcetype="B",null(),kb)
| eval sessionId=if(sourcetype="A",null(),sessionId)
| stats sum(kb) dc(sessionid) by ip
```

splunk> .conf19

© 2019 SPLUNK INC.

What about streamstats and eventstats?

Yes. Super powerful. Super useful.

However despite its name streamstats is not a "distributable streaming" command.

So while many transaction use cases that aren't simple "by id" transactions can be refactored to use a combination of eval and streamstats + stats, you'll still be breaking MapReduce.   You might be better off sticking with transaction.

Test it both ways !!

© 2019 SPLUNK INC.

Trick – walk softly and carry a big transforming command

When you have a big expression with 2 or more transforming commands, try and make the first one do most of the work reducing the number of rows.

Sometimes you can "set the table" really well with eval and streaming commands such that one big stats command can work a miracle in one pass, and thus do it out at the indexers too.

```
| eval {type}_duration=duration
| eval {type}_callId=callId
| `crazypants_macro_to_calculate_and_mvexpand_name_and_number_fields`
| stats dc(incoming_callId) as incoming dc(outgoing_callId) as outgoing
  dc(internal_callId)  as internal dc(callId) as total
  sum(incoming_duration) as incoming_duration sum(outgoing_duration) as
  sum(duration) as total_duration values(name) as name by number
```

© 2019 SPLUNK INC.

Trick – break it into two problems.

Sometimes when there's just way too much going on,   look for a way to break it into two problems where you can bake one of them out as a lookup.

Look at what pieces only change rarely. Imagine if THOSE get baked out daily as a lookup, does the rest of the problem get easier?

Simple example -- I want to know the phones that have NOT made a call in the last week (and have thus generated no data)   I could do a search over all time, then join with the same search over the last week.

Better - make a lookup that represents "all phones that have ever been seen" (ie with that expensive all time search). Then:

```
<terms> | eval present=1| inputlookup all_phones_ever append=t
| stats values(present) as present by extension
| search NOT present=1
```

splunk> .conf19

© 2019 SPLUNK INC.

If there's a way, you can find it.

Or at least…. you can find crazy people in the community who will help you find it.

Even in super complex reporting situations, even after you've beaten it to death and given up, there are strange helpful people on Slack/Answers who have arcane knowledge and can totally help you.

The #search-help and #tinfoilstats channels on Slack are your friends.

Make sure you give sufficient details and this generally means posting the SPL (yes, the raw actual Version not a simplified version) [redacted]

Thanks to everyone who makes these Splunk Community things happen! ! ! !

Please send any and all feedback or thoughts to
nick@sideviewapps.com

© 2019 SPLUNK INC.

.conf19

splunk>

Thank You!

Go to the .conf19 mobile app to

RATE THIS SESSION

© 2019 SPLUNK INC.

Hey you clicked past the fake ending slide!!

Nice work

.conf19
splunk>

© 2019 SPLUNK INC.

Glossary

Here are those official Splunk docs about grouping data from different sources (the page with the flow chart).
https://docs.splunk.com/Documentation/Splunk/7.3.1/Search/Abouteventcorrelation
https://docs.splunk.com/Documentation/Splunk/7.3.1/Search/Abouttransactions

Subsearch: technically this refers to SPL expressions in square brackets, in a plain old search clause.
https://docs.splunk.com/Documentation/Splunk/7.3.1/Search/Aboutsubsearches
And conversely the term does NOT refer to how square brackets are used by the join or append commands.

Finalizing - is basically when you click the "stop" button on a running search. Splunk stops getting new events off disk and kind of "wraps up" and returns the partial results. It might look complete but it's not.
Autofinalizing - is when this is done automatically (and quietly) on a search or subsearch.

Join/append – You can read this but… pretty much every time it tells you join or append is OK, it's wrong.
https://docs.splunk.com/Documentation/Splunk/7.3.1/SearchReference/SQLtoSplunk
So remember I said that. Note I avoided saying "inner left" join or any such sqlism here in this talk. They can all be done. If you were hoping for a mapping of how to do each one with stats and eval, I am sorry to disappoint.

Disjunction - is just a fancy word for using "OR" to join two sets of searchterms together.

splunk> .conf19

© 2019 SPLUNK INC.

Problem – I need more… "distinctness".

In this example, we have “OrderNo”, and then “start” and “end” that are both times. The need was to calculate for each Service, the average time elapsed per order. The trick was that there were often numerous transactions per OrderNo and we had to treat each separately when calculating averages.

We relied on an assumption – that the Orders would never have interleaved transactions, and we used streamstats to supply the extra “distinctness”.

```
<search string>
| streamstats dc(start_time) as transaction_count by OrderNo
| stats earliest(start_time) as start_time earliest(stop_time) as
stop_time by OrderNo, transaction_count, Service
| eval duration=tostring(stop_time-start_time)
| stats mean(duration) as avg_duration by Service
| table Service, avg_duration
```

© 2019 SPLUNK INC.

Problem – I have gaps in my ids

I don't really have one good id – instead I have two bad ones. Each side of the data has its own.
Luckily, there are some events that have both.

This feels a lot like a transaction use case and it might be.
(it might even be a searchtxn use case).

But whenever you have to kind of "fill in" or "smear out" data across a bunch of rows, also think of eventstats and streamstats.

Here we use eventstats to smear the JSESSIONID values over onto the other side.

```
sourcetype=access_combined OR sourcetype=appserver_log
| eventstats values(JSESSIONID) as JSESSIONID by ecid
| stats avg(foo) sum(bar) values(baz) by JSESSIONID
```

splunk> .conf19

© 2019 SPLUNK INC.

Problem – I need the raw event data

And with transaction I get the _raw for free.

Do you really?   I mean both "need" it and get it "for free" ?

But for debugging, yes absolutely the _raw can be super useful cause transaction keeps you in the "events' view.   However you can get some mileage out of

```
... | stats list(_raw) as events by someId
```

Or even this technique, which can wedge ANY transforming result back into the "events" view. [redacted]

```
Foo NOT foo
| append [
  search SEARCHTERMS | stats count sum(kb) as kb list(_raw) as
_raw by clientip host]
```

splunk> .conf19

© 2019 SPLUNK INC.

Problem – I need to search 2 different timeranges

But the two sides have different timeranges so I need join/append.

I need to see, out of the users active in the last 24 hours, the one with the highest number of incidents over the last 30 days.

```
  sourcetype=A | stats count by userid               (last 24 hours)
  sourcetype=B | stats dc(incidentId) by userid   (Last 7 days)
```

First back up – is the big one so static it could be a lookup?

```
  sourcetype=B | stats dc(incidentId) by userid | outputlookup user_incidents_7d.csv
```

OR Is the second one so small and cheap that it could be a simple subsearch?

```
  sourcetype=B [search sourcetype=A earliest=-24h@h | stats count by userid | fields
userid ]
  | stats dc(incidentId) by userid
```

© 2019 SPLUNK INC.

Problem – I need to search 2 different timeranges

Nice try but that wont work

No and the final results need to end up with values from that "inner" search, so I can't use a subsearch.

```
sourcetype=A | stats count values(host) by userid (-24h)
sourcetype=B | stats dc(incidentId) by userid     (-7d)
```

No problem. Stats.

```
  sourcetype=A OR sourcetype=B
| eval isRecentA=
   if(sourcetype=A AND _time>relative_time(now(), "-24h@h"),1,0)
| where sourcetype=B OR isRecentA=1
| eval hostFromA=if(sourcetype=A,host,null())
| stats dc(incidentId) values(hostFromA) as hosts by userid
```

© 2019 SPLUNK INC.

Problem – I need to search 2 different timeranges

But…

But that search…

```
  sourcetype=A OR sourcetype=B
| eval isRecentA=
 if(sourcetype=A AND _time>relative_time(now(),
"-24h@h"),1,0)
| where sourcetype=B OR isRecentA=1
| eval hostFromA=if(sourcetype=A,host,null())
| stats dc(incidentId) values(hostFromA) as hosts by userid
```

… it gets lots of A off disk and then throws it away

True. it's out at the indexers at least!   "At least make the hot air go far away?".
But yes, the corresponding join may indeed be less evil here.   Test it!

© 2019 SPLUNK INC.

A random point out on the long tail

A fun jaunt with chart, stats and xyseries, eval stats and… ok I lost count.

Sorry smart guy, I literally need to join the result output of two *different* transforming commands.

```
sourcetype=A | chart count over userid by app

sourcetype=B | stats sum(kb) by userid
```

For each user I need the eventcounts across the 5 apps, PLUS the total KB added up from sourcetype B.   I need stats behavior AND I need chart behavior!

Therefore I need join!

© 2019 SPLUNK INC.

A random point out on the long tail

Nope. Stats.   You may have heard that "chart is just stats wearing a funny hat". Or if you haven't heard that before, well you've heard it now.

Step #1) You can always refactor chart into a stats and an xyseries

```
  | chart count over userid by application
```

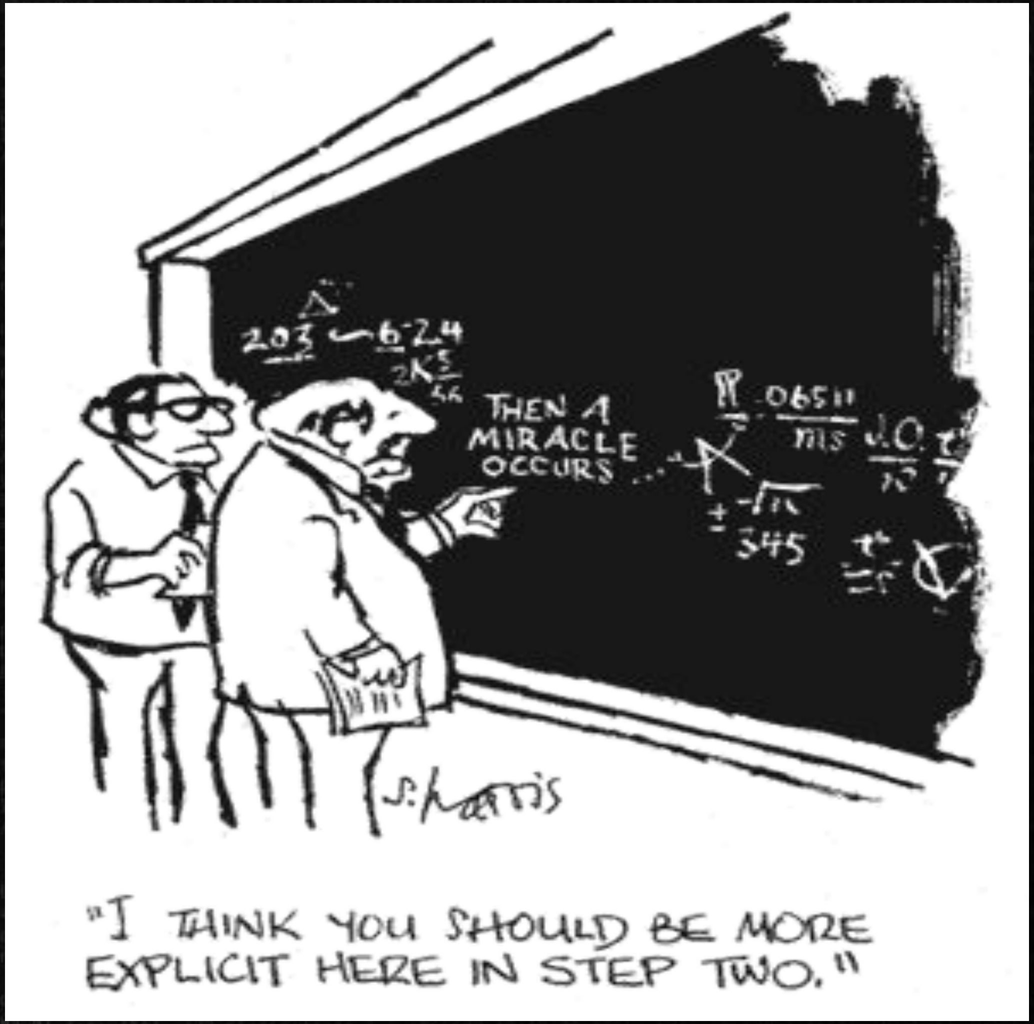
Is equivalent to

```
  | stats count by userid application
  | xyseries userid application count
```

© 2019 SPLUNK INC.

A random point out on the long tail

ok I'm bluffing. I'm not going to walk you through all this cause it's a little insane.


THEN A MIRACLE OCCURS . . .
S. Harris
"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

splunk> .conf19

© 2019 SPLUNK INC.

A random point out on the long tail

but here we are, having combined that chart AND the stats into a single expression.

```
sourcetype=A OR sourcetype=B
| fillnull application value="NULL"
| stats sum(kb) as kb count by userid application
| eval application=if(application="NULL",null(),application)
| eval clown_car = userid + ":::" + kb
| chart count over clown_car by application
| eval clown_car=mvsplit(clown_car, ":::")
| eval userid=mvindex(clown_car, 0)
| eval kb=mvindex(clown_car, 1)
| table userid kb *
```