Master Joining Datasets Without Using Join

How to Build Amazing Reports Across Multiple Datasets Without Sacrificing Performance

Nick Mealy, CEO
Sideview LLC

https://sideviewapps.com

October 2018

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.

NICK MEALY

CEO, Sideview LLC

nick@sideviewapps.com

Former Splunk Mad Scientist 2005-2010

sideview / madscient on answers/IRC

Why is this guy qualified to give this talk?

Former Splunk Mad Scientist and Principal UI Developer 2005 -2010

Rob and I in the booth at LinuxWorld in 2005 when we launched Splunk 1.0.

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.

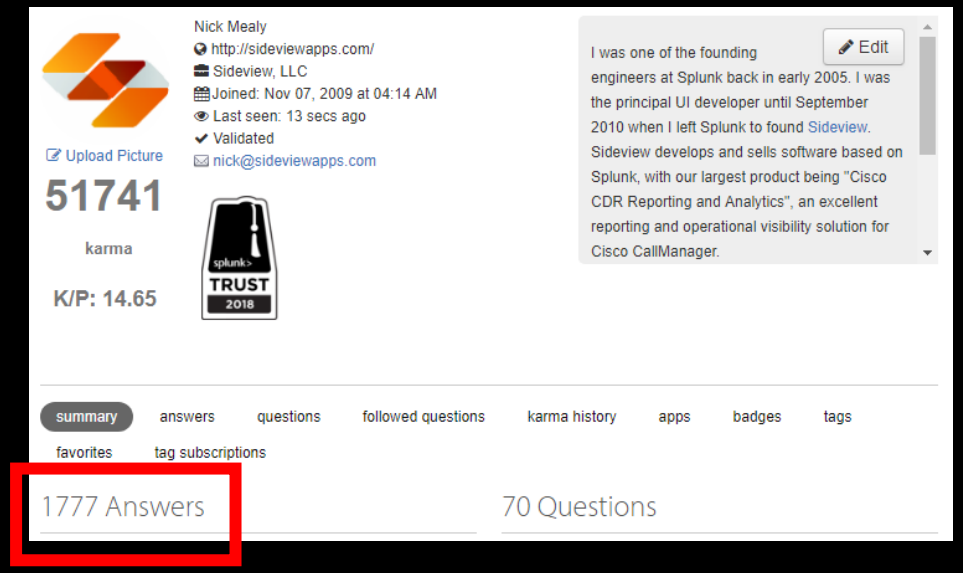
The first people to hit the bottom learn how to welcome everyone else.

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

Only two people vomited that I know of.

OK so… 2010. How about the last 8 years?

For many years until about a year ago I was one of the folks who would 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

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.

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

Let's take a quick poll.

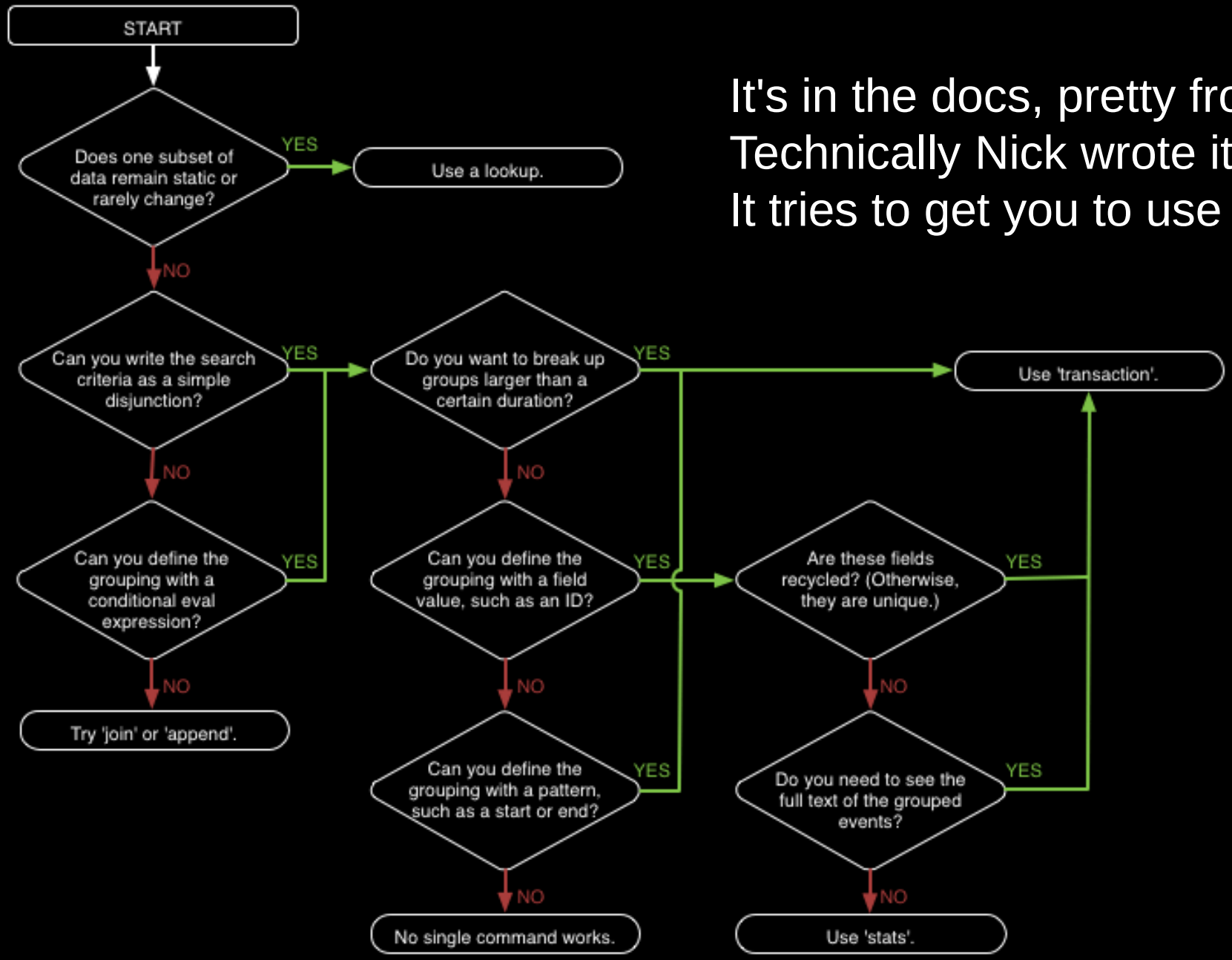
How many of you:

1) personally run fairly expensive searches on your Splunk instances that use join/append or transaction?

2) administer Splunk instances where searches like that are running?

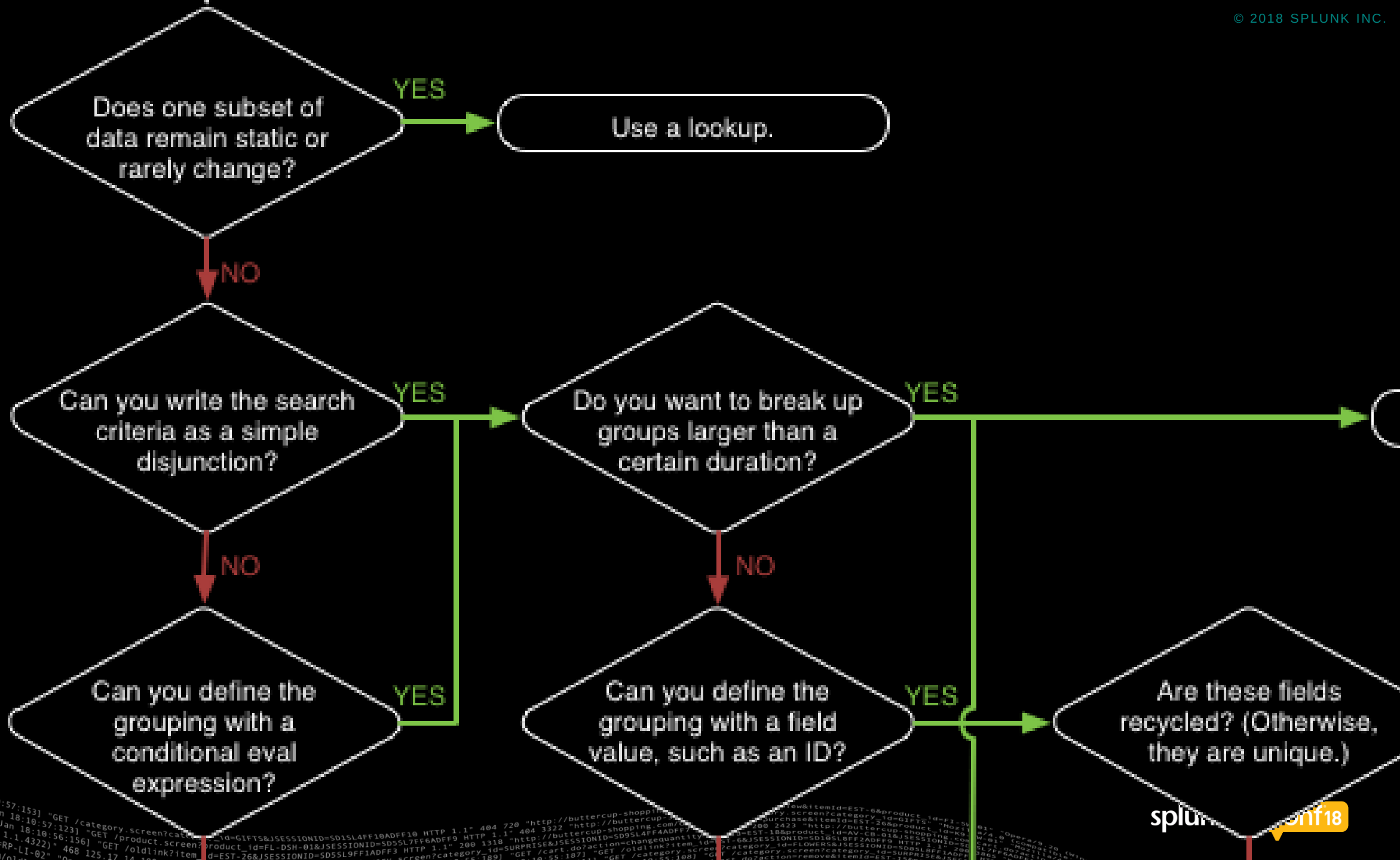
3) have the feeling these searches might be 2x or 10x faster if rewritten more cleverly?

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?



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> .conf18



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.

- truncation if you exceed 50,000 rows. (and/or oom anxiety if you mess with limits.conf)
- Autofinalized when execution time exceeds 120 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 and processing back to the SH. Forcing Splunk to do all statistical work on the SH.**
- **As a kind of “worst-practice”, it proliferates quickly.**

What's wrong with the transaction command?

- It's designed for edge cases - keeping all the arguments straight can be hard.
- **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 almost certainly 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

"omg it completes in 20 minutes now".

MapReduce – How Splunk's implementation works.

Say we want to see call durations for 1000 devices, across a million calls.

We want to push as much work as we can out to the indexers..

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

Well, we definitely send this part out…

But if that's ALL we do, it's really wasteful. All million rows come back to the SH and all the math has to happen on the SH.

Whereas all we NEED from each node is the total duration they see for the 100 devices.

MapReduce – How Splunk's implementation works.

"prestats" is how each indexer sends back only "sufficient statistics".

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

"distributable streaming portion"
Will include evals, rex, where etc..
Indexers run this part PLUS prestats

"Transforming portion"

From the first non-"distributable streaming" command all the way to the end.

SH runs this at the end to tie it all together.

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

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
```

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
| 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

Quick Demo in Browser

Show a search that has search + eval + fields + stats + fillnull.

Show Job Inspector, how to see what is sent to indexers

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

Show sistats output in browser as an example of what sufficient statistics look like.

Show those same rows being happily accepted as input rows by stats.

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!

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.

Accidentally circumventing MapReduce entirely

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”.

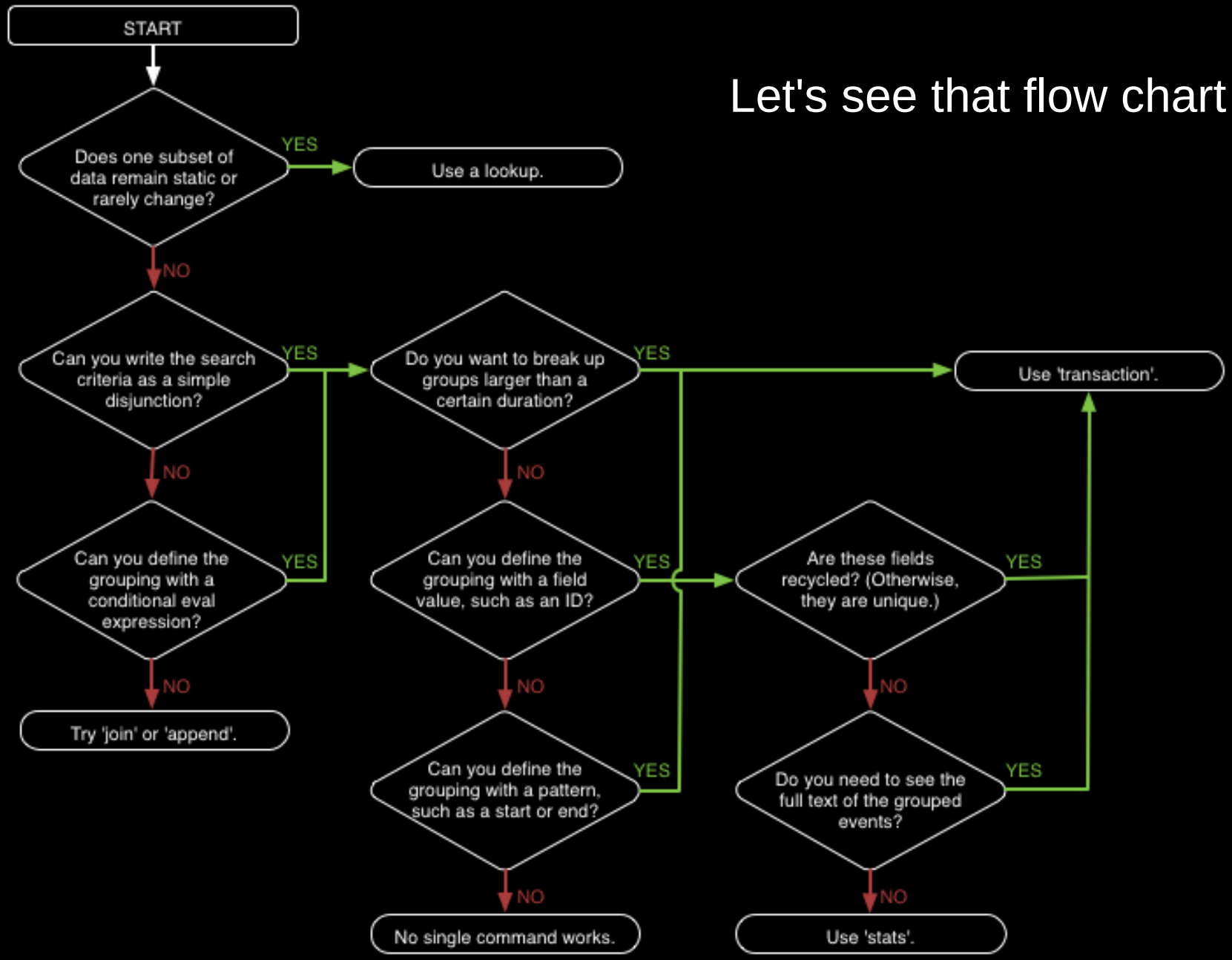
docs are using this word “transaction”. -- "oh hey there's a transaction command".

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

NO. Lookups and the core commands should be your first tools.

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

Let's see that flow chart again.



splunk> .conf18

Sure, I know. “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.

Part 1 Conclusion - surface roads are a last resort

splunk> .conf18

Wait wait – isn't there a new thing in 7.1?

Yes! 7.1 introduced the "redistribute" command, aka "parallel reduce"

Now that you know what MapReduce does, you can understand what it does.

Redistribute gives you a very advanced way to do Map + Reduce + Reduce.

Ie it can just for a particular search, briefly deputize a subset of your indexers into "intermediate reducers".

The bad news -- it's not really for you if…

- if you don't have lots of indexers.
- if your indexers are already overloaded (it *adds* net indexer load)
- if your SPL still has room for improvement in the main "MapReduce" arena.
- if you use multi-site clustering

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
```

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
```

Example #2

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

AWFUL:

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

15.03 seconds

Nope – just use stats twice.

Good:

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

1.76 seconds

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

Example #3

normalizing field names

… because 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.

Example #4

Normalizing field values

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

```
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.

Example #5

Gluing things to other things

…. because I just need to glue some result rows together ($boss wants it this way). :|

```
sourcetype=A | stats avg(session_length) as length
  +
sourcetype=B | stats dc(sessions) as sessions dc(users) as users
  =
sourcetype=A | stats avg(session_length) as length
| appendcols [search sourcetype=B | stats dc(sessions) as sessions
dc(users) as users]
```

Example #5

Gluing things to other things, continued

…. because I just need to glue some result rows together ($boss wants it this way). :)

```
sourcetype=A | stats avg(session_length) as length
  +
sourcetype=B | stats dc(sessions) as sessions dc(users) as users
  =
sourcetype=A OR sourcetype=B
| stats avg(session_length) as length dc(sessions) as sessions dc(users)
```

Stats doesn't care here – it can calculate both of them just fine and for each calculation it throws away any null values.

Likewise multivalues – it can be a surprise that stats will handle multivalued fields perfectly well.

Example #6

Gluing + joinery

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> .conf18

Example #6

Gluing + joinery, continued

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
```

Example #7 – Timerange shenanigans

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
```

splunk> .conf18

Example #8 – Timerange shenanigans

Nice try but that wont work.

No and I 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
```

splunk> .conf18

Example #8 – Timerange shenanigans

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!

Example #8 – Timerange shenanigans

You silly – you can combine earliest/latest in your parens now!

You can! But unfortunately it doesn't do what you think it does.

```
  ( sourcetype=A earliest=-24h@h latest=now) OR
   ( sourcetype=B earliest=-7d@d latest=now)
| eval hostFromA=if(sourcetype=A,host,null())
| stats dc(incidentId) values(hostFromA) as hosts by userid
```

It really feels like this is going to be smart and get only 24h of A off disk but it doesn't. It gets 7d@d of A off disk out at the indexers and then postfilters.

So it isn't any less evil than the previous slide, just less explicit.

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 !!

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 values(loginUserID) as loginUserID values(huntPilotDN) as huntPilotDN
 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 outgoing_duration sum(internal_duration) as
 internal_duration sum(duration) as total_duration values(partyName) as name by number
```

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

Even in mind-blowingly complex reporting situations, there are strange helpful people on Slack/Answers who have arcane knowledge and can totally help you.

MAKE SURE YOU GIVE SUFFICIENT DETAILS, and post the SPL you have.

Also #tinfoilstats channel on Slack

Please send any and all feedback or thoughts to

nick@sideviewapps.com

Thank You

Don't forget to rate this session in the .conf18 mobile app

.conf18

splunk>

You're still here?

Hey – you clicked past the fake ending slide.

Nice work.

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.

"What pieces only change rarely, and if I imagine THOSE at least are all getting baked out as a lookup, does the rest get easier."

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
```

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
```

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 (It might even be a searchtxn use case) and it might be.

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
```

Example #9 – I need the raw event data

Do you really?

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

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

Or even this, which will shove ANY transforming result back to the "events" view.

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

Example #10 – I just need to define transactions by ID

continued

A SPL magic trick:

```
| transaction field1 field2
| stats sum(foo) count by field1 field2
```

gives the same results as:

```
| stats sum(foo) count by field1 field2
```

Example #11 – long tail of advanced cases

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!

Example #11 – long tail of advanced cases

Nope. Stats. I mention this more to emphasize that there are strange helpful people on Slack/Answers who have arcane knowledge and can totally help you.

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

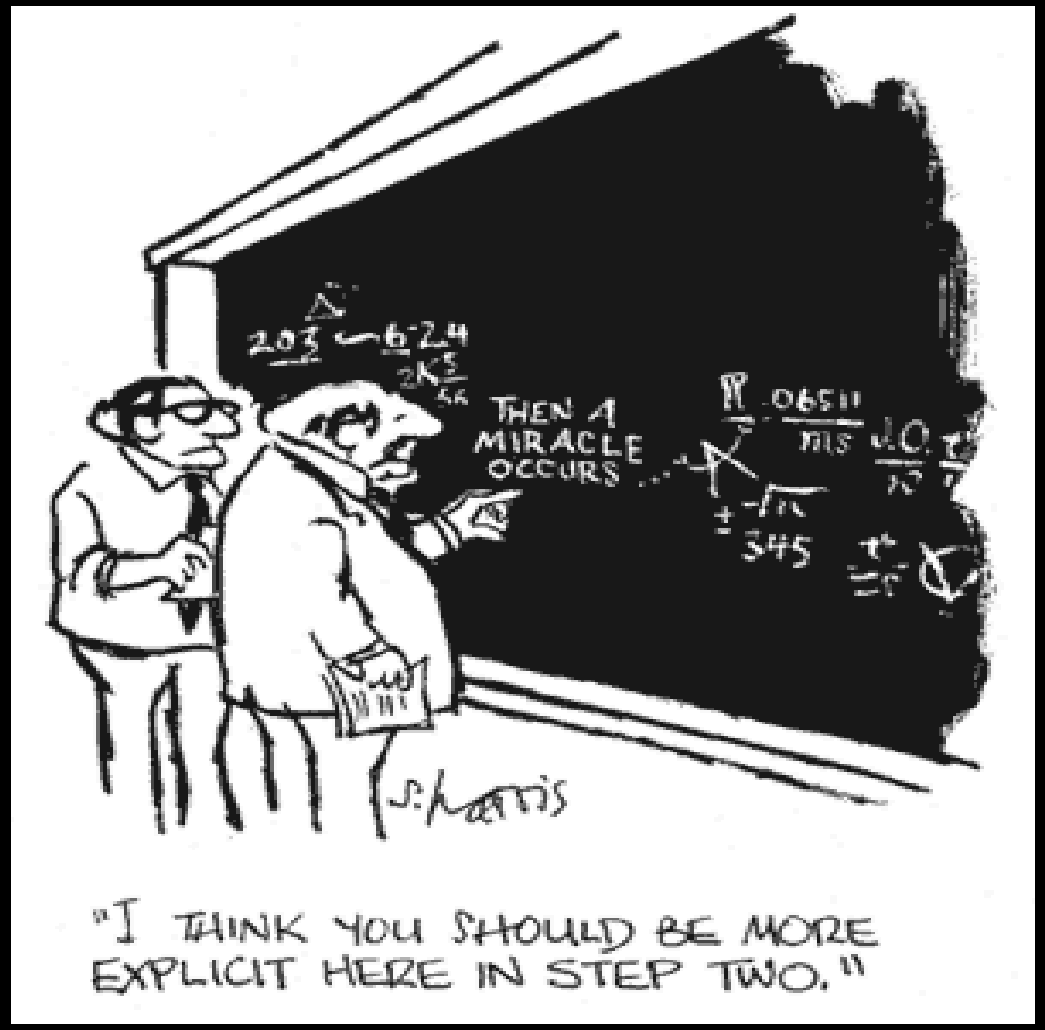
```
| chart count over userid by application
```

Is equivalent to

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

Example #11 – long tail of advanced cases

Step #2 ok I'm bluffing. I'm not going to walk you through all this cause it's insane.



Example #11 – long tail of advanced cases

```
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 workaround=userid + ":::" + kb
| chart count over workaround by application
| eval workaround =mvsplit(workaround,":::")
| eval userid=mvindex(workaround,0)
| eval kb=mvindex(workaround,1)
| table userid kb *
```