

# Master Joining Datasets Without Using Join

Nick Mealy



CEO, Chief Mad Scientist | Sideview, LLC



# Forward-Looking Statements



During the course of this presentation, we may make forward-looking statements regarding future events or plans 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 may differ materially. The forward-looking statements made in the this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, it may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements made herein.

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 functionalities described or to include any such feature or functionality in a future release.

Splunk, Splunk>, Data-to-Everything, D2E and Turn Data Into Doing 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. © 2020 Splunk Inc. All rights reserved





# Nick Mealy

CEO, Chief Mad Scientist | Sideview, LLC



# Why Are We Here?

- You maybe have a bunch of important searches that use join, append or appendcols.
- You might have been told that join/append are bad. You might be irritated because they still seem totally necessary.
- However there's a nagging suspicion you have, that maybe the eggheads are right, and you can rewrite these searches to be better.
- And that possibly then you could greatly reduce resource load on your search heads (And make the searches faster too).

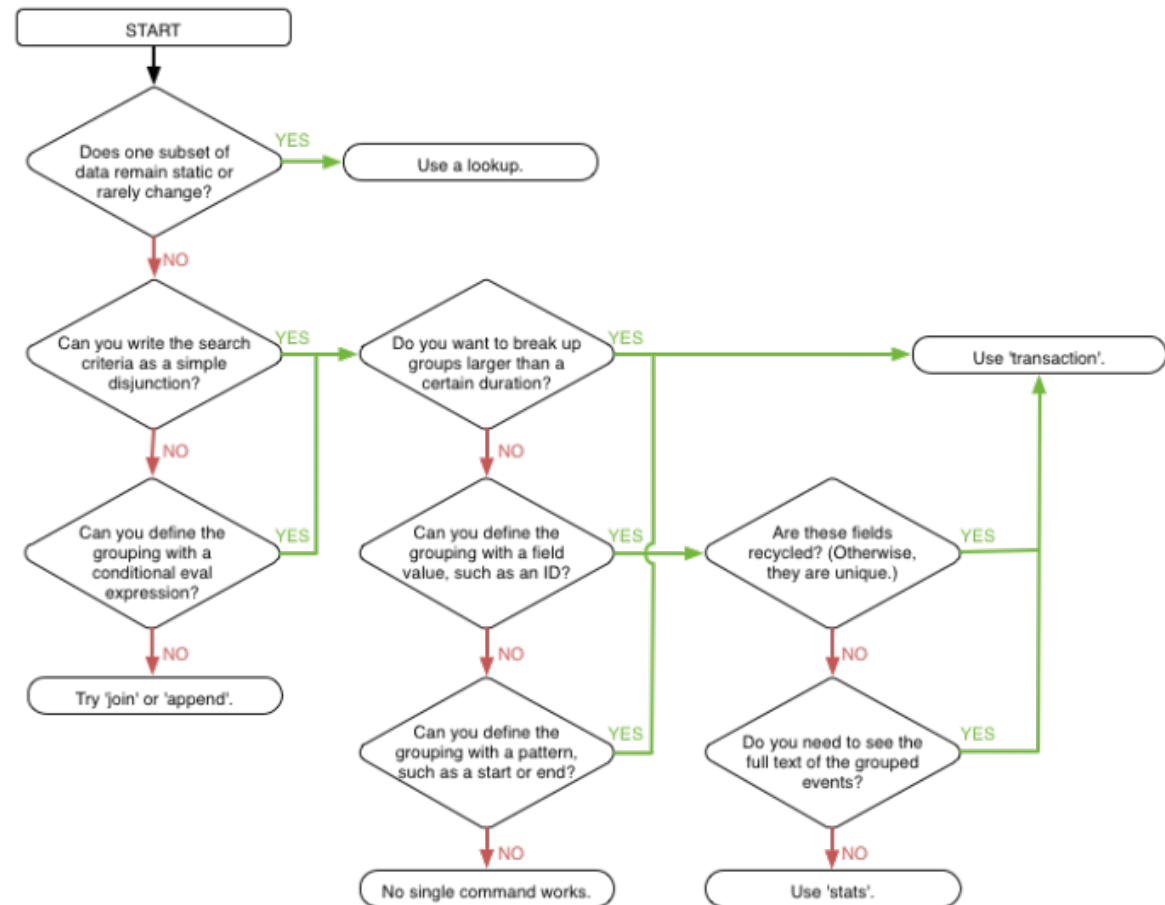
# Why Is This Guy Qualified to Give This Talk?

- Former Splunk Mad Scientist and Principal UI Developer 2005-2010
- This is Rob Das and I in the booth at Linuxworld in 2005, the week we all launched Splunk 1.0.



# OK, So... I Heard 2010. How About the Last 10 Years?

- I was the karma leader on Answers once upon a time, and it was in large part from answering search language questions in this particular area.
- Also the docs for this stuff still uses a flowchart I made ages ago.





# OK, So... 2016? Keep Going...

- I never stopped working with Splunk fulltime in a technical role and our main product for Cisco CallManager has to do some pretty hairy SPL.

```
`cdr_events' duration>0
| eval {type}_duration=duration
| eval {type}_callId=callId
| eval {bh_type}_duration=duration
| eval {bh_type}_callId=callId
| eval {callMediaType}_duration=duration
| eval {callMediaType}_callId=callId
| `calculate_all_internal_parties`
| 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
sum(off_hours_duration) as off_hours_duration sum(business_hours_duration) as business_hours_duration dc
  (business_hours_callId) as business_hours dc(off_hours_callId) as off_hours sum(audio_duration) as
  audio_duration sum(video_duration) as video_duration dc(audio_callId) as audio dc(video_callId) as video
  values(partyName) as name by number
| `fields_for_internal_parties`
```

# The Talk

assuming we ever get to it

... what's it like?

## 1) **Why are the bad things bad?**

Why join and append are evil.

(Why transaction is chaotic neutral)

## 2) **How to be good. How to 'use stats' even when it really looks like you can't.**

You may get a tendency to say "I wonder if we can use some conditional eval to fix this."

## 3) **Do I have to? Are my searches really bad?**

(iow is the SH stuck doing most of the work?)



# The Talk

The tl;dr version

## 1) Join and append really don't scale.

They're designed to be last resorts for weird cases. They underuse the IDX tier and overload the SH. raising limits.conf keys only helps a little.

## 2) You use lots of OR and lots of eval in your SPL.

Sourcetype=A OR sourcetype=B

You send a terrible ugly heap of different events down the pipeline and let eval/stats and friends sort them out for you. It's crazy unintuitive.

## 3) The Job Inspector is your friend.

And avoid the table command, unless you need to break mapreduce and thus measure how much work it was doing before



# Part 1: Why Are the Bad Things Bad?

---



# Naming Things – It's Hard.

The names are really unfortunate and send users off 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.

# What's Wrong With the Join and Append Commands?

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

- truncation if you exceed 50,000 rows. (and/or oom anxiety)
- Your job is quietly autofinalized (another kind of truncation) if you exceed execution time.
- 2 jobs instead of 1.
- Potentially getting the same data off disk twice.
- You might not even *\*realize\** that you're hitting autofinalize and row-truncation limits, but they are nonetheless making your results wrong.
- Breaking Map/Reduce, OR making it less efficient. I.e. forcing splunk to pull more data and processing back to the search head.
- As a kind of “worst-practice”, it proliferates quickly.



# Search Head Says It's Tired – You Do It.

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

- YOU ARE NOW THE SEARCH HEAD!
- You don't know the best way to do this.
- But... what's the **worst** way?
- Let's have indexers send us **ALL** events  
**We'll** store them all  
**We'll** do all the search filtering locally  
**We'll** add up all the durations!!!!

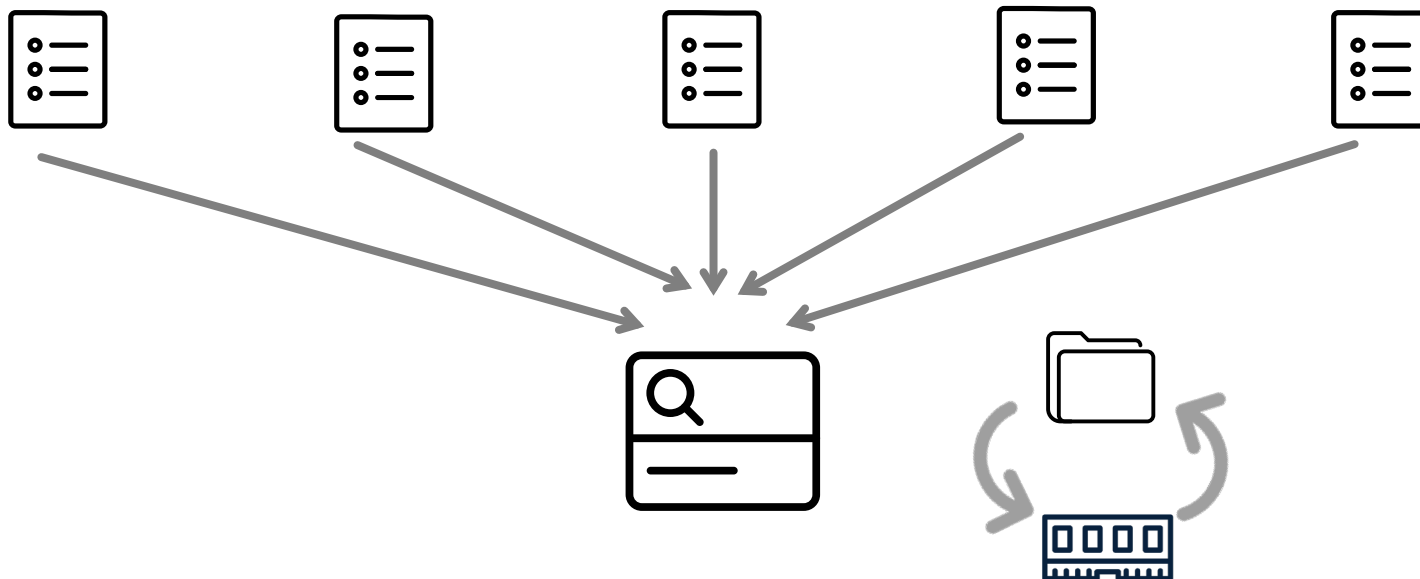


# OK That Was Awful.

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

network: So you like packets eh?

Filesystem/RAM – pulling everything to one host, there's just WAY too much data!



## Is preview on?

Then the UI adds another layer of hellish load by forcing a finalize every few seconds.

# Next – Lets Try Something a Little Less Terrible

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



- So let's at least send this part out. Then the indexers can send only the matching events over the wire.
- MUCH better. We're still getting an ungodly number of rows and doing a lot of math, so it still sucks but we are now at least doing “**distributed search**”.
- Still a MASSIVE load on the SH's disk I/O, RAM and CPU though.

# Search Head Says It's Tired – You Do It

At least that was doing “distributed search. Let’s do “distributed reporting” too.

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



What if there was a way we could kind of... make the indexers do ALL the work.

Like, what if each indexer only sent back a tiny little table, literally like this:

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




# And That's What Happens

- It's how Splunk does not just “distributed search” but “distributed reporting”.
- It's how you generally scale by adding indexers – the SH tier doesn't (shouldn't) have much work.
- It's the “Reduce” part of splunk's “Map Reduce” implementation. Or at least a really big part of it.
- These little tables are sometimes called the “sufficient statistics” – half-baked cakes cooked by the indexers.

# How It's Actually Implemented = “pre commands”

```
sourcetype=cdr type=outgoing
```



## 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
```

```
stats sum(duration) by device_type
```



## Transforming Portion

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

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

# And This Has All Been Happening Automatically!





# Unless... You've Been Inadvertently Writing Suboptimal SPL.





# ... Like That Ever Happens.





# Part 2: How to Be Good

How to 'use stats' even when it feels like  
you can't



# Example 1

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

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

+

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

=

**Awful**

```
sourcetype=db  
| join pid [  
    search sourcetype=app  
| 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

**Awful**

```
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

Just send the whole mess along to stats/chart/timechart.

**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

... because 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
```

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
```

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

because the field names are different, but when I use rename it damages the “good” one.

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.<sup>1</sup>

# Example #4

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

```
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.





# Part 3: Do I have to?

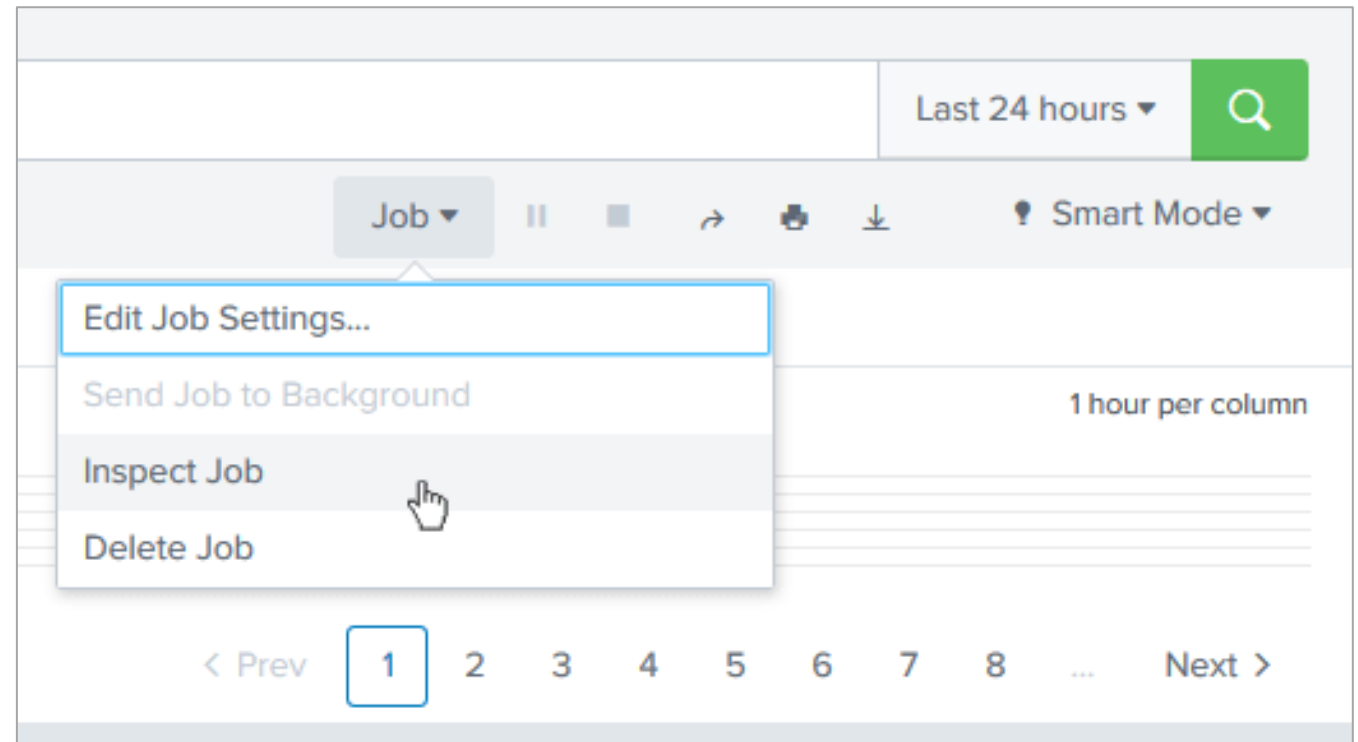
---

How bad are my searches, really?



# How to know how you're doing (review)?

- 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



# Do I Have to? Maybe Not!

**Small data , fast searches, rows<<50K – none of this really matters. Pick your battles**

**High cardinality “by id” searches– because of the nature of the query, sometimes even prestats is outputting a huge number of rows too. This cuts down on potential resources saved at the SH.**

**I only have a couple indexers anyway.**

**Search Heads are free to me because someone else maintains them ← wait NO don't be evil!**

# What About Streamstats and Eventstats?

- Yes. Super powerful. Super useful. They deserve a whole talk to themselves.
- However despite its name streamstats is not a "distributable streaming" command, nor is eventstats and therefore they force the data to come back to the Search Head there.
- 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 rework it so that the first stats command will do most of the work reducing the weight involved.
- Sometimes you can eliminate the second one entirely.
- 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
```

# Trick – Break It Into Two Problems

- Sometimes when there's just way too much going on.
- 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 one hugely expensive all time search). Then bake that out as a lookup, then:

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

# 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 there to help.
- Make sure you give sufficient details and this generally means posting the SPL (yes, the raw actual
- Version not a simplified version)
- Thanks to everyone who made Virtual Conf happen, who made the SplunkTrust conf track happen!
- And thanks for watching/reading/listening!
- Please send any and all feedback or thoughts to
- [nick@sideviewapps.com](mailto:nick@sideviewapps.com)
- And please do take the session survey.





**Hey!**  
**You clicked past the  
fake ending slide!**

**Nice work**

# 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/8.0.5/Search/Abouteventcorrelation>
- <https://docs.splunk.com/Documentation/Splunk/8.0.5/Search/Abouttransactions>
- **Subsearch**: technically this refers to SPL expressions in square brackets, in a plain old search clause.
- <https://docs.splunk.com/Documentation/Splunk/8.0.5/Search/Aboutsubsearches>
- And conversely the term imo should 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.
- **Disjunction** - is just a fancy word for using "OR" to join two sets of searchterms together.
- 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/8.0.5/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.

# 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 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
```

# 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 debugging mileage out of:

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

- Or even this trick, which forces ANY transforming result back into the "events" tab:

Foo NOT foo

```
| append [search SEARCHTERMS | stats count sum(kb) as kb list(_raw) as _raw by  
clientip host]
```



# Problem – I Need To Search Two 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
```

# Problem – I Need To Search Two 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. Still 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
```

# Problem – I Need To Search Two 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!

# Problem – I Need Chart To Mix Split-by Columns With Normal Columns

Aka “clown car techniques”

- I need to have rows that are users, and then some columns that are from a “split by” field expression, and some other columns that are NOT from that splitby.
- Sometimes this is expressed as needing to join the result output of two \*different\* transforming commands, ie:

```
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. This totally feels like what join is made for. 1

# Clown Car, Continued

- 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.
- It turns out that 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
```



# Clown Car, Continued

It's a little awful – you mash all the values into one clown-car field, and group by that, then unpack them all from the clown car afterward.  
Here is a SIMPLE example.

```
sourcetype=A OR sourcetype=B
| stats sum(kb) as kb count by userid 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 *
```