📊 Analytics & Dashboards

> **Series:** OPLOGS | **Notebook:** 7 of 8 | **Created:** December 2025

Aggregation, Time Series, and Visualization Queries

This notebook covers aggregation functions, time series analysis, statistical patterns, and dashboard-ready queries for log analytics.

Table of Contents

Prerequisites

- ✅ Access to a Dynatrace environment with log data
- ✅ Completed OPLOGS-01 through OPLOGS-06
- ✅ Familiarity with dashboard concepts

1. Aggregation Functions

```
pbmVhckdyYWRpZW50IGlkPSJ0aW1lR3JhZEFnZyIgeDE9IjAlIiB5MT0iMCUiIHgyPSIxMDAlIiB5
Mj0iMTAwJSI+CiAgICAgIDxzdG9wIG9mZnNldD0iMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiNmNTllM
GI7c3RvcC1vcGFjaXR5OjEiIC8+CiAgICAgIDxzdG9wIG9mZnNldD0iMTAwJSIgc3R5bGU9InN0b3
AtY29sb3I6I2Q5NzcwNjtzdG9wLW9wYWNpdHk6MSIgLz4KICAgIDwvbGluZWFyR3JhZGllbnQ+CiA
gICA8ZmlsdGVyIGlkPSJhZ2dTaGFkb3ciPgogICAgICA8ZmVEcm9wU2hhZG93IGR4PSIxIiBkeT0i
MSIgc3RkRGV2aWF0aW9uPSIyIiBmbG9vZC1vcGFjaXR5PSIwLjE1Ii8+CiAgICA8L2ZpbHRlcj4KI
CA8L2RlZnM+CgogIDwhLS0gQmFja2dyb3VuZCAtLT4KICA8cmVjdCB3aWR0aD0iODAwIiBoZWlnaH
Q9IjMyMCIgZmlsbD0iI2Y4ZjlmYSIgcng9IjEwIi8+CgogIDwhLS0gVGl0bGUgLS0+CiAgPHRleHQ
geD0iNDAwIiB5PSIyOCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9
IjE4IiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0iIzMzMyIgdGV4dC1hbmNob3I9Im1pZGRsZSI+
RFMIEFnZ3JlZ2F0aW9uIEZ1bmN0aW9ucyBSZWZlcmVuY2U8L3RleHQ+CiAgPHRleHQgeD0iNDAwIi
B5PSI0OCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjExIiBmaWx
sPSIjNjY2IiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5Vc2Ugd2l0aCBzdW1tYXJpemUgY29tbWFuZDog
c3VtbWFyaXplHtyZXN1bHQgPSBmdW5jdGlvbigpfSwgYnk6e2ZpZWxkfTwvdGV4dD4KICAgPCEtL
SBDb3VudGluZyBGdW5jdGlvbnMgLS0+CiAgPHJlY3QgeD0iMzAiIHk9IjcwIiB3aWR0aD0iMTc1Ii
BoZWlnaHQ9IjIzMCIgcng9IjEwIiBmaWxsPSIjZmZmIiBzdHJva2U9IiMzYjgyZjYiIHN0cm9rZS1
3aWR0aD0iMiIvPgogIDxyZWN0IHg9IjMwIiB5PSIzMCIgd2lkdGg9IjE3NSIgaGVpZ2h0PSIzNSIg
cng9IjEwIiBmaWxsPSJ1cmwoI2NvdW50aW5nR3JhZCkiLz4KICA8dGV4dCB4PSIxMTciIHk9Ijk1I
iBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTIiIGZvbnQtd2VpZ2
h0PSJib2xkIiBmaWxsPSJ3aGl0ZSIgdGV4dC1hbmNob3I9Im1pZGRsZSI+Q291bnRpbmc8L3RleHQ
+CgogIDx0ZXh0IHg9IjUwIiB5PSIxMjUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6
ZT0iMTEiIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSIjMjU2M2ViIj5jb3VudCgpPC90ZXh0PgogI
Dx0ZXh0IHg9IjUwIiB5PSIxNDAiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC
1zaXplPSIxMCIgZmlsbD0iIzY0NzQ4YiI+VG90YWwgcmVjb3JkczwvdGV4dD4KICAgPHRleHQgeD0
iNTAiIHk9IjE2NSIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMSIgZm9udC13
ZWlnaHQ9ImJvbGQiIGZpbGw9IiMyNTYzZWIiPmNvdW50SWYoY29uZCk8L3RleHQ+CiAgPHRleHQgeD
0iNTAiIHk9IjE4MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9Ij
EwIiBmaWxsPSIjNjQ3NDhiIj5Db3VudCB3aGVyZSBjb25kaXRpb248L3RleHQ+CgogIDx0ZXh0IHg
9IjUwIiB5PSIyMDUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTEiIGZvbnQt
d2VpZ2h0PSJib2xkIiBmaWxsPSIjMjU2M2ViIj5jb3VudERpc3RpbmN0KGYpPC90ZXh0PgogIDx0Z
Xh0IHg9IjUwIiB5PSIyMjAiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaX
plPSIxMCIgZmlsbD0iIzY0NzQ4YiI+VW5pcXVlIHZhbHVlczwvdGV4dD4KICA8cmVjdCB4PSIzMCI
gaWR0aD0iMTc1IiBoZWlnaHQ9IjM1IiByeD0iMTAiIGZpbGw9InVybCgjc3RhdHNHcmFkKSIvPgog
IDx0ZXh0IHg9IjExNyIgeT0iMjQiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iM
TIiIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSIjZmZmIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5TdG
F0aXN0aWNhbDwvdGV4dD4KICA8dGV4dCB4PSIxMTciIHk9IjM4IiBmb250LWZhbWlseT0iQXJpYWw
sIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTEiIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSJ3aGl0
ZSIgdGV4dC1hbmNob3I9Im1pZGRsZSI+Q291bnRpbmc8L3RleHQ+CgogIDx0ZXh0IHg9IjUwIiB5P
SIxMDAiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTEiIGZvbnQtd2VpZ2h0P
SJib2xkIiBmaWxsPSIjMDU5NjY5Ij5zdW0oZmllbGQpPC90ZXh0PgogIDx0ZXh0IHg9IjUwIiB5PS
IxMDUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0
iIzY0NzQ4YiI+VW5pcXVlIHdoZXJlIGNvbmQ8L3RleHQ+CgogIDxyZWN0IHg9IjMwIiB5PSIxMz
UiIGhlaWdodD0iMzUiIHJ4PSIxMCIgZmlsbD0idXJsKCNjc3RhdHNHcmFkKSIvPgogIDx0ZXh0IHg
9IjExNyIgeT0iMTU4IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjExIiBmb250
LXdlaWdodD0iYm9sZCIgZmlsbD0iI2ZmZiIgdGV4dC1hbmNob3I9Im1pZGRsZSI+QXZnKGZpZWxkK
TwvdGV4dD4KICA8dGV4dCB4PSIxMTciIHk9IjE3MiIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLX
NlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjZmZmIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5Db3V
udERpc3RpbmN0PC90ZXh0PgogIDx0ZXh0IHg9IjUwIiB5PSIxOTUiIGZvbnQtZmFtaWx5PSJtb2
5vc3BhY2UiIGZvbnQtc2l6ZT0iMTEiIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSIjMDU5NjY5Ij
5zdW0oZmllbGQpPC90ZXh0PgogIDx0ZXh0IHg9IjUwIiB5PSIyMDUiIGZvbnQtZmFtaWx5PSJBc
mlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzY0NzQ4YiI+VW5pcXVlIHdoZX
JlIGNvbmQ8L3RleHQ+CgogIDxyZWN0IHg9IjMwIiB5PSIyMjUiIGhlaWdodD0iMzUiIHJ4PSIxM
CIgZmlsbD0idXJsKCNjc3RhdHNHcmFkKSIvPgogIDx0ZXh0IHg9IjExNyIgeT0iMjQ1IiBmb250LW
ZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjExIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0
iI2ZmZiIgdGV4dC1hbmNob3I9Im1pZGRsZSI+TWluKGZpZWxkKTwvdGV4dD4KICA8dGV4dCB4PSIx
MTciIHk9IjI2MiIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwI
iBmaWxsPSIjZmZmIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5Db3VudERpc3RpbmN0PC90ZXh0PgogIC
A8dGV4dCB4PSI1MCIgeT0iMjg1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjE
xIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0iIzA1OTY2OSI+c3VtKGZpZWxkKTwvdGV4dD4KICA8
dGV4dCB4PSI1MCIgeT0iMzA1IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc
2l6ZT0iMTAiIGZpbGw9IiM2NDc0OGIiPlVuaXF1ZSB3aGVyZSBjb25kPC90ZXh0PgogIDx0ZXh0IH
g9IjUwIiB5PSIzMjUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSI
xMCIgZmlsbD0iIzY0NzQ4YiI+VW5pcXVlIHdoZXJlIGNvbmQ8L3RleHQ+CiAgPHRleHQgeD0iNT
AiIHk9IjM0NSIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiB
maWxsPSIjNjQ3NDhiIj5EaXN0aW5jdCBjb3VudERpc3RpbmN0PC90ZXh0PgogIDx0ZXh0IHg9IjUw
IiB5PSIzNjUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZ
mlsbD0iIzY0NzQ4YiI+VW5pcXVlIHZhbHVlczwvdGV4dD4KICA8dGV4dCB4PSI1MCIgeT0iMzg1Ii
BmaWxsPSIjNjQ3NDhiIj5Db3VudERpc3RpbmN0PC90ZXh0PgogIDx0ZXh0IHg9IjUwIiB5PSI0M
DUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIz
Y0NzQ4YiI+VW5pcXVlIHdoZXJlIGNvbmQ8L3RleHQ+CiAgPHRleHQgeD0iNTAiIHk9IjQyNSIgZ
m9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjNjQ3ND
hiIj5EaXN0aW5jdCBjb3VudERpc3RpbmN0PC90ZXh0PgogIDx0ZXh0IHg9IjUwIiB5PSI0NDUiI
GZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzY0Nz
Q4YiI+VW5pcXVlIHZhbHVlczwvdGV4dD4KICA8dGV4dCB4PSI1MCIgeT0iNDY1IiBmb250LWZhbWl
seT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM2NDc0OGIiPlVuaX
F1ZSB3aGVyZSBjb25kPC90ZXh0PgogIDx0ZXh0IHg9IjUwIiB5PSI0ODUiIGZvbnQtZmFtaWx5P
SJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzY0NzQ4YiI+VW5pcXVlIH
doZXJlIGNvbmQ8L3RleHQ+CiAgPHRleHQgeD0iNTAiIHk9IjUwNSIgZm9udC1mYW1pbHk9IkFyaWF
sLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjNjQ3NDhiIj5VbmlxdWUgd2hlcmUg
Y29uZDwvdGV4dD4KICA8dGV4dCB4PSI1MCIgeT0iNTI1IiBmb250LWZhbWlseT0iQXJpYWwsIHNhb
nMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM2NDc0OGIiPlVuaXF1ZSB3aGVyZSBjb25k
```

hbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM2NDc0OGIiPlRv
dGFsIG9mIHZhbHVlczwvdGV4dD4KCiAgPHRleHQgeD0iMjQwIiB5PSIxNjUiIGZvbnQtZmFtaWx5P
SJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTEiIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSIjMDU5Nj
Y5Ij5hdmcoZmllbGQpPC90ZXh0PgogIDx0ZXh0IHg9IjI0MCIgeT0iMTgwIiBmb250LWZhbWlseT0
iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM2NDc0OGIiPkF2ZXJhZ2Ug
dmFsdWU8L3RleHQ+CgogIDx0ZXh0IHg9IjI0MCIgeT0iMjA1IiBmb250LWZhbWlseT0ibW9ub3NwY
WNlIiBmb250LXNpemU9IjExIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0iIzA1OTY2OSI+bWluKG
ZpZWxkKTwvdGV4dD4KICA8dGV4dCB4PSIyNDAiIHk9IjIyMCIgZm9udC1mYW1pbHk9IkFyaWFsLCB
zYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjNjQ3NDhiIj5NaW5pbXVtIHZhbHVlPC90
ZXh0PgoKICA8dGV4dCB4PSIyNDAiIHk9IjI0NSIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9ud
C1zaXplPSIxMSIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IiMwNTk2NjkiPm1heChmaWVsZCk8L3
RleHQ+CiAgPHRleHQgeD0iMjQwIiB5PSIyNjAiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJ
pZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzY0NzQ4YiI+TWF4aW11bSB2YWx1ZTwvdGV4dD4KCiAg
PHJlY3QgeD0iMjQwIiB5PSIyNzUiIHdpZHRoPSIxMzUiIGhlaWdodD0iMTgiIHJ4PSI0IiBmaWxsP
SIjZDFmYWU1Ii8+CiAgPHRleHQgeD0iMzA3IiB5PSIyODgiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2
Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzA0Nzg1NyIgdGV4dC1hbmNob3I9Im1pZGR
sZSI+UmV0dXJuczogZG91YmxlPC90ZXh0PgoKICA8IS0tIENvbGxlY3Rpb24gRnVuY3Rpb25zIC0t
PgogIDxyZWN0IHg9IjQxMCIgeT0iNzAiIHdpZHRoPSIxNzUiIGhlaWdodD0iMjMwIiByeD0iMTAiI
GZpbGw9IiNmZmYiIHN0cm9rZT0iIzhiNWNmNiIgc3Ryb2tlLXdpZHRoPSIyIi8+CiAgPHJlY3QgeD
0iNDEwIiB5PSI3MCIgd2lkdGg9IjE3NSIgaGVpZ2h0PSIzNSIgcng9IjEwIiBmaWxsPSJ1cmwoI2N
vbGxlY3Rpb25HcmFkKSIvPgogIDx0ZXh0IHg9IjQ5NyIgeT0iOTUiIGZvbnQtZmFtaWx5PSJBcmlh
bCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMiIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9Indoa
XRlIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5Db2xsZWN0aW9uPC90ZXh0PgoKICA8dGV4dCB4PSI0Mz
AiIHk9IjEyNSIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMSIgZm9udC13ZWl
naHQ9ImJvbGQiIGZpbGw9IiM3YzNhZWQiPmNvbGxlY3RBcnJheShmKTwvdGV4dD4KICA8dGV4dCB4
PSI0MzAiIHk9IjE0MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9I
jEwIiBmaWxsPSIjNjQ3NDhiIj5BbGwgdmFsdWVzIGFzIGFycmF5PC90ZXh0PgoKICA8dGV4dCB4PS
I0MzAiIHk9IjE2NSIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMSIgZm9udC1
3ZWlnaHQ9ImJvbGQiIGZpbGw9IiM3YzNhZWQiPmNvbGxlY3REaXN0aW5jdChmKTwvdGV4dD4KICA8
dGV4dCB4PSI0MzAiIHk9IjE4MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250L
XNpemU9IjEwIiBmaWxsPSIjNjQ3NDhiIj5VbmlxdWUgdmFsdWVzIGFycmF5PC90ZXh0PgoKICA8dG
V4dCB4PSI0MzAiIHk9IjIwNSIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMSI
gZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IiM3YzNhZWQiPnRha2VGaXJzdChmKTwvdGV4dD4KICA8
dGV4dCB4PSI0MzAiIHk9IjIyMCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250L
XNpemU9IjEwIiBmaWxsPSIjNjQ3NDhiIj5GaXJzdCB2YWx1ZSBpbiBncm91cDwvdGV4dD4KCiAgPH
RleHQgeD0iNDMwIiB5PSIyNDUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTE
iIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSIjN2MzYWVkIj50YWtlTGFzdChmKTwvdGV4dD4KICA8
dGV4dCB4PSI0MzAiIHk9IjI2MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250L
XNpemU9IjEwIiBmaWxsPSIjNjQ3NDhiIj5MYXN0IHZhbHVlIGluIGdyb3VwPC90ZXh0PgoKICA8cm
VjdCB4PSI0MzAiIHk9IjI3NSIgd2lkdGg9IjEzNSIgaGVpZ2h0PSIxOCIgcng9IjQiIGZpbGw9IiN
lZGU5ZmUiLz4KICA8dGV4dCB4PSI0OTciIHk9IjI4OCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5z
LXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjNmQyOGQ5IiB0ZXh0LWFuY2hvcj0ibWlkZGxlI
j5SZXR1cm5zOiBhcnJheTwvdGV4dD4KCiAgPCEtLSBUaW1lIEZ1bmN0aW9ucyAtLT4KICA8cmVjdC
B4PSI2MDAiIHk9IjcwIiB3aWR0aD0iMTcwIiBoZWlnaHQ9IjIzMCIgcng9IjEwIiBmaWxsPSIjZmZ
mIiBzdHJva2U9IiNmNTllMGIiIHN0cm9rZS13aWR0aD0iMiIvPgogIDxyZWN0IHg9IjYwMCIgeT0i
NzAiIHdpZHRoPSIxNzAiIGhlaWdodD0iMzUiIHJ4PSIxMCIgZmlsbD0idXJsKCN0aW1lR3JhZEFnZ
ykiLz4KICA8dGV4dCB4PSI2ODUiIHk9Ijk1IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaW
YiIGZvbnQtc2l6ZT0iMTIiIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSJ3aGl0ZSIgdGV4dC1hbmN

```
ob3I9Im1pZGRsZSI+UGVyY2VudGlsZTwvdGV4dD4KCiAgPHRleHQgeD0iNjIwIiB5PSIxMjUiIGZv
bnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTEiIGZvbnQtd2VpZ2h0PSJib2xkIiBma
WxsPSIjZDk3NzA2Ij5wZXJjZW50aWxlKGYsNTApPC90ZXh0PgogIDx0ZXh0IHg9IjYyMCIgeT0iMT
QwIiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM
2NDc0OGIiPk1lZGlhbiAocDUwKTwvdGV4dD4KICAgPHRleHQgeD0iNjIwIiB5PSIxNjUiIGZvbnQt
ZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTEiIGZvbnQtd2VpZ2h0PSJib2xkIiBma
WxsPSIjZDk3NzA2Ij5wZXJjZW50aWxlKGYsOTUpPC90ZXh0PgogIDx0ZXh0IHg9IjYyMCIgeT0iMTgwIi
Bmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM2NDc
0OGIiPjk1dGggcGVyY2VudGlsZTwvdGV4dD4KICAgPHRleHQgeD0iNjIwIiB5PSIyMDUiIGZvbnQt
ZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTEiIGZvbnQtd2VpZ2h0PSJib2xkIiBma
WxsPSIjZDk3NzA2Ij5wZXJjZW50aWxlKGYsOTkpPC90ZXh0PgogIDx0ZXh0IHg9IjYyMCIgeT0iMjIwIi
Bmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM2NDc
0OGIiPjk5dGggcGVyY2VudGlsZTwvdGV4dD4KICAgPHRleHQgeD0iNjIwIiB5PSIyNDUiIGZvbnQt
ZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTEiIGZvbnQtd2VpZ2h0PSJib2xkIiBma
WxsPSIjZDk3NzA2Ij5zdGRkZXYoZmllbGQpPC90ZXh0PgogIDx0ZXh0IHg9IjYyMCIgeT0iMjYwIiBmb2
50LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM2NDc0OGI
iPlN0YW5kYXJkIGRldmlhdGlvbjwvdGV4dD4KICAgPHRleHQgeD0iNjIwIiB5PSIyNzUiIHdpZHRo
PSIxMzAiIGhlaWdodD0iMTgiIHJ4PSI0IiBmaWxsPSIjZmVmM2M3Ii8+CiAgPHRleHQgeD0iNjIwIiB5
PSIyODgiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZm
lsbD0iIzkyNDAwZSIgdGV4dC1hbmNob3I9Im1pZGRsZSI+UmV0dXJuczogZG91YmxlPC90ZXh0Pgo
8L3N2Zz4K)



### Common Aggregation Functions

| Function | Description | Example |
|----------|-------------|---------|
| `count()` | Count records | `summarize {total = count()}` |
| `countIf(condition)` | Conditional count | `countIf(loglevel == "ERROR")` |
| `countDistinct(field)` | Unique values | `countDistinct(k8s.pod.name)` |
| `sum(field)` | Sum values | `sum(bytes)` |
| `avg(field)` | Average | `avg(response_time)` |
| `min(field)` | Minimum | `min(timestamp)` |
| `max(field)` | Maximum | `max(timestamp)` |
| `takeFirst(field)` | First value | `takeFirst(content)` |
| `takeLast(field)` | Last value | `takeLast(content)` |
| `collectDistinct(field)` | Array of unique values |
`collectDistinct(loglevel)` |

### 🧯 Important: Named Aliases Required

```dql
// ✅ CORRECT - Named aliases
| summarize {count = count(), errors = countIf(...)}, by: {...}
| sort count desc

// ❌ WRONG - Anonymous aggregation can't be used in sort
```
```

```
| summarize count(), by: {...}
| sort count() desc  // ERROR!
```

```python
// Basic aggregations
fetch logs, from: now() - 1h
| summarize {
    total_logs = count(),
    unique_hosts = countDistinct(dt.entity.host),
    unique_namespaces = countDistinct(k8s.namespace.name),
    unique_pods = countDistinct(k8s.pod.name)
  }
```

```python
// Aggregations with grouping
fetch logs, from: now() - 1h
| summarize {
    total = count(),
    errors = countIf(loglevel == "ERROR"),
    warnings = countIf(loglevel == "WARN"),
    info = countIf(loglevel == "INFO")
  }, by: {k8s.namespace.name}
| sort total desc
| limit 15
```

```python
// Error rate calculation
fetch logs, from: now() - 1h
| summarize {
    total = count(),
    errors = countIf(loglevel == "ERROR")
  }, by: {k8s.namespace.name}
| filter total > 100  // Minimum sample size
| fieldsAdd error_rate_pct = round((errors * 100.0) / total, decimals: 2)
| sort error_rate_pct desc
| limit 15
```

2. Time Series Analysis

The `makeTimeseries` command creates time-bucketed data for trend visualization.

```python
// Log volume over time (5-minute buckets)
```

```
fetch logs, from: now() - 6h
| makeTimeseries {log_count = count()}, interval: 5m
```

```python
// Error trend over time
fetch logs, from: now() - 6h
| makeTimeseries {
    total = count(),
    errors = countIf(loglevel == "ERROR"),
    warnings = countIf(loglevel == "WARN")
  }, interval: 5m
```

```python
// Time series by dimension (namespace)
fetch logs, from: now() - 6h
| filter isNotNull(k8s.namespace.name)
| makeTimeseries {
    log_count = count()
  }, by: {k8s.namespace.name}, interval: 10m
```

```python
// Error rate time series by host
fetch logs, from: now() - 6h
| filter isNotNull(dt.entity.host)
| makeTimeseries {
    total = count(),
    errors = countIf(loglevel == "ERROR")
  }, by: {dt.entity.host}, interval: 15m
```

3. Statistical Analysis

```python
// Log volume statistics by source
fetch logs, from: now() - 24h
| summarize {
    total = count(),
    first_seen = min(timestamp),
    last_seen = max(timestamp)
  }, by: {dt.openpipeline.source}
| fieldsAdd duration_hours = (last_seen - first_seen) / 3600000000000
| fieldsAdd logs_per_hour = round(total / duration_hours, decimals: 0)
| sort total desc
```

```python
// Percentile analysis (if numeric field available)
fetch logs, from: now() - 1h
| fieldsAdd content_length = stringLength(content)
| summarize {
    avg_length = avg(content_length),
    min_length = min(content_length),
    max_length = max(content_length),
    total_logs = count()
  }, by: {k8s.namespace.name}
| sort avg_length desc
| limit 15
```

```python
// Hourly distribution analysis
fetch logs, from: now() - 24h
| fieldsAdd hour_bucket = bin(timestamp, 1h)
| summarize {log_count = count()}, by: {hour_bucket}
| sort hour_bucket asc
```

```python
// Daily distribution analysis
fetch logs, from: now() - 7d
| fieldsAdd day_bucket = bin(timestamp, 1d)
| summarize {
    log_count = count(),
    error_count = countIf(loglevel == "ERROR")
  }, by: {day_bucket}
| sort day_bucket asc
```

4. Dashboard-Ready Queries

These queries are optimized for dashboard tiles and visualizations.

```python
// Single Value: Total Logs (Last Hour)
fetch logs, from: now() - 1h
| summarize {total_logs = count()}
```

```python
// Single Value: Error Count (Last Hour)
fetch logs, from: now() - 1h
| filter loglevel == "ERROR"
| summarize {error_count = count()}
```

```

```python
// Pie Chart: Log Level Distribution
fetch logs, from: now() - 1h
| summarize {count = count()}, by: {loglevel}
| sort count desc
```

```python
// Bar Chart: Top Error Sources
fetch logs, from: now() - 1h
| filter loglevel == "ERROR"
| summarize {error_count = count()}, by: {k8s.namespace.name}
| sort error_count desc
| limit 10
```

```python
// Line Chart: Log Trend (6 Hours)
fetch logs, from: now() - 6h
| makeTimeseries {
    errors = countIf(loglevel == "ERROR"),
    warnings = countIf(loglevel == "WARN"),
    info = countIf(loglevel == "INFO")
  }, interval: 5m
```

```python
// Table: Top Error Messages
fetch logs, from: now() - 1h
| filter loglevel == "ERROR"
| fieldsAdd error_preview = substring(content, from: 0, to: 100)
| summarize {
    occurrences = count(),
    first_seen = min(timestamp),
    last_seen = max(timestamp)
  }, by: {error_preview, k8s.namespace.name}
| sort occurrences desc
| limit 20
```

```python
// Heat Map: Errors by Hour and Namespace
fetch logs, from: now() - 24h
| filter loglevel == "ERROR"
| fieldsAdd hour_bucket = bin(timestamp, 1h)
| summarize {error_count = count()}, by: {hour_bucket, k8s.namespace.name}
```
```

```
| sort hour_bucket asc
```

5. Trend Analysis

```python
// Compare current hour to previous hour
fetch logs, from: now() - 2h
| fieldsAdd period = if(timestamp > now() - 1h, "current", else: "previous")
| summarize {count = count()}, by: {period, loglevel}
| sort period asc, count desc
```

```python
// New error patterns (appeared in last hour)
fetch logs, from: now() - 1h
| filter loglevel == "ERROR"
| fieldsAdd error_sig = substring(content, from: 0, to: 80)
| summarize {
    count = count(),
    first_seen = min(timestamp)
  }, by: {error_sig, k8s.namespace.name}
| filter first_seen > now() - 30m  // First seen in last 30 minutes
| sort first_seen desc
| limit 15
```

```python
// Volume anomaly detection (compare to baseline)
fetch logs, from: now() - 6h
| fieldsAdd time_bucket = bin(timestamp, 15m)
| summarize {log_count = count()}, by: {time_bucket, k8s.namespace.name}
| sort time_bucket desc
| limit 100
```

6. Log Pattern Analysis

```python
// Top log patterns (by content prefix)
fetch logs, from: now() - 1h
| fieldsAdd pattern = substring(content, from: 0, to: 60)
| summarize {count = count()}, by: {pattern}
| sort count desc
| limit 25
```

```python
```

```
// Unique log levels and statuses
fetch logs, from: now() - 1h
| summarize {
    loglevels = collectDistinct(loglevel),
    statuses = collectDistinct(status),
    sources = collectDistinct(dt.openpipeline.source)
  }
```

```python
// Log diversity score (unique patterns per namespace)
fetch logs, from: now() - 1h
| fieldsAdd pattern = substring(content, from: 0, to: 50)
| summarize {
    total_logs = count(),
    unique_patterns = countDistinct(pattern)
  }, by: {k8s.namespace.name}
| fieldsAdd diversity_ratio = round((unique_patterns * 100.0) / total_logs,
decimals: 2)
| sort diversity_ratio desc
| limit 15
```

7. Operational Dashboard Queries

```python
// Operations Summary (last 1h)
fetch logs, from: now() - 1h
| summarize {
    total_logs = count(),
    error_count = countIf(loglevel == "ERROR"),
    warn_count = countIf(loglevel == "WARN"),
    unique_hosts = countDistinct(dt.entity.host),
    unique_pods = countDistinct(k8s.pod.name)
  }
| fieldsAdd error_rate = round((error_count * 100.0) / total_logs, decimals:
2)
```

```python
// Health scorecard by namespace
fetch logs, from: now() - 1h
| filter isNotNull(k8s.namespace.name)
| summarize {
    total = count(),
    errors = countIf(loglevel == "ERROR"),
    warnings = countIf(loglevel == "WARN")
  }, by: {k8s.namespace.name}
```

```
| fieldsAdd error_rate = round((errors * 100.0) / total, decimals: 2)
| fieldsAdd health_status = if(error_rate > 10, "CRITICAL",
                            else: if(error_rate > 5, "WARNING",
                            else: "HEALTHY"))
| sort error_rate desc
```

```python
// Ingestion pipeline summary
fetch logs, from: now() - 1h
| summarize {
    log_count = count(),
    unique_pipelines = countDistinct(dt.openpipeline.pipelines)
  }, by: {dt.openpipeline.source, dt.system.bucket}
| sort log_count desc
```

📝 Summary

In this notebook, you learned:

✅ **Aggregation functions** - count, countIf, countDistinct, sum, avg
✅ **Time series** - makeTimeseries with intervals
✅ **Statistical analysis** - hourly/daily patterns, percentiles
✅ **Dashboard queries** - Single values, charts, tables
✅ **Trend analysis** - Period comparison, anomaly detection
✅ **Pattern analysis** - Log clustering and diversity

➡️ Next Steps

Continue to **OPLOGS-08: Security & Data Protection** for data protection patterns.

📚 References

- [DQL Aggregation Functions](https://docs.dynatrace.com/docs/platform/grail/dynatrace-query-language/functions/aggregation-functions)
- [DQL makeTimeseries](https://docs.dynatrace.com/docs/platform/grail/dynatrace-query-language/commands/makeTimeseries)

```
- [Dynatrace Dashboards](https://docs.dynatrace.com/docs/observe-and-
explore/dashboards-and-notebooks/dashboards-new)
```