

## # 🔎 Querying & Parsing Logs

> \*\*Series:\*\* OPL0GS | \*\*Notebook:\*\* 5 of 8 | \*\*Created:\*\* December 2025

### ## DQL Fundamentals and DPL Pattern Matching

This notebook covers DQL query syntax, filtering, string matching, and DPL (Dynatrace Pattern Language) for extracting structured data from logs.

---

### ## Table of Contents

1. DQL Fundamentals
2. Filtering Logs
3. String Matching Functions
4. Field Selection and Transformation
5. DPL (Dynatrace Pattern Language) Parsing
6. Working with JSON Logs
7. Null Handling
8. Advanced Parsing Examples

### ## Prerequisites

- Access to a Dynatrace environment with log data
- Completed OPL0GS-01 through OPL0GS-04
- Basic understanding of regular expressions (helpful)

### ## 1. DQL Fundamentals

#### ### Query Structure

```
! [DQL Pipeline Model]
(
ciIHZpZXdB3g9IjAgMCA4MDAgMzAwIj4KICAgIDxsaw5lYXJHcmFkaWVudCBpZD0
iZmV0Y2hHcmFkIiB4MT0iMCUiIHkxPSIwJSIgeDI9IjEwMCUiIHkyPSIxMDALIj4KICAgICA
b3Agb2Zmc2V0PSIwJSIgc3R5bGU9InN0b3AtY29sb3I6IzNi0DJmNjtzdG9wLw9wYwNpdHk6MSIg
Lz4KICAgICA
PHN0b3Agb2Zmc2V0PSIxMDALIiBzdHlsZT0ic3RvcC1jb2xvcjojMjU2M2Vi03N0b3
Atb3BhY2l0eToxIiAvPgogICA
PC9saW5lYXJHcmFkaWVudD4KICAgIDxsaw5lYXJHcmFkaWVudCB
pZD0iZmlsdGVyR3JhZERRTCIgeDE9IjAlIiB5MT0iMCUiIHgyPSIxMDALIiB5Mj0iMTAwJSI+CiAg
ICA
IDxdG9wIG9mZnNldD0iMCUiIH
N0eWxlPSJzdG9wLwNvbG9y0iNlZjQ0NDQ7c3RvcC1vcGFja
XR50jEiIC8+CiAgICA
IDxdG9wIG9mZnNldD0iMTAwJSIgc3R5bGU9InN0b3AtY29sb3I6I2RjMj
YyNjtzdG9wLw9wYwNpdHk6MSIg
Lz4KICAgIDwvbGluZW
FyR3JhZG1lbnQ+CiAgICA
8bGluZW
FyR3J
hZG1lbnQgaWQ9ImZpZwxc0dyYWQiIHgxPSIxJSIgeTE9IjAlIiB4Mj0iMTAwJSIgeTI9IjEwMCUi
```

PgogICAgICA8c3RvcCBvZmZzZXQ9IjAlIIiBzdHlsZT0ic3RvcC1jb2xvcjojMTBi0Tgx03N0b3Atb3BhY2l0eToxiAvPgogICAgICA8c3RvcCBvZmZzZXQ9IjEwMCUiIHN0eWxlPSJzdG9wLWNvbG9y0iMwNTk2Njk7c3RvcC1vcGFjaXR50jEiIC8+CiAgICA8L2xpbmVhckdyYWRpZW50PgogICAgPGxpbmVhckdyYWRpZW50IGlkPSJzdW1tYXJpemVHcmFkIiB4MT0iMCUiIHkxPSIwJSIgeDI9IjEwMCUiIHkyPSIxMDA1Ij4KICAgICAgPHN0b3Agb2Zmc2V0PSIwJSIgc3R5bGU9InN0b3AtY29sb3I6IzhINWNmNjtzdG9wLW9wYWNPdHk6MSIgLz4KICAgICAgPHN0b3Agb2Zmc2V0PSIxMDA1IiBzdHlsZT0ic3RvcC1jb2xvcjojN2MzYWV0k03N0b3Atb3BhY2l0eToxiAvPgogICAgPC9saW5LYXJHcmFkaWVudD4KICAgIDxsaw5LYXJHcmFkaWVudCBpZD0ic29ydEdyYWQiIHgxPSIwJSIgeTE9IjAlIIiB4Mj0iMTAwJSIgeTI9IjEwMCUiPgogICAgICA8c3RvcCBvZmZzZXQ9IjAlIIiBzdHlsZT0ic3RvcC1jb2xvcjojZjU5ZTBi03N0b3Atb3BhY2l0eToxiAvPgogICAgICA8c3RvcCBvZmZzZXQ9IjEwMCUiIHN0eWxlPSJzdG9wLWNvbG9y0iNk0Tc3MDY7c3RvcC1vcGFjaXR50jEiIC8+CiAgICA8L2xpbmVhckdyYWRpZW50PgoPgICAgPGZpbHRlcipBpZD0iZHFsU2hhZG93Ij4KICAgICAgPGZlRHJvcfNoYWVrdyBkeD0iMSIgZHk9IjEiIHN0ZERldmlhdGlvbj0iMiIgZmxvb20tb3BhY2l0eT0iMC4xNSIVPgogICAgPC9maWx0ZXI+CiAgICA8bWFya2VyIGlkPSJkcWxBcnJvdyIgbWFya2VyV21kdGg9IjEwIIiBtYXJrZXJIZWlnaHQ9IjciIHJlZlg9IjkiIHJlZlk9IjMuNSIgB3JpZW50PSJhdXRvIj4KICAgICAgPHBvbHlnb24gcG9pbnRzPSIxIDA5IDEwIDMuNSwgMCA3IiBmaWxsPSIjNjQ3NDhiIi8+CiAgICA8L21hcmtlcj4KICAgICA8L2R1ZnM+CgogIDwhLS0gQmFja2dyb3VuZCATLT4KICAgICA8cmVjdCB3aWR0aD0iODAwIIBoZWlnaHQ9IjMwMCiGZmlsbD0iI2Y4ZjlmYSIgcn9IjEwIIi8+CgogIDwhLS0gVGlobGUgLS0+CiAgPHRleHQgeD0iNDAwIIiB5PSIy0CIgZm9udC1mYW1pbhk9IkFyaWfsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjE4IIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0iIzMzMyIgdGV4dC1hbmnob3I9Im1pZGRsZSI+RFFMIFF1ZXJ5IFBpcGVsaW5lIE1vZGVsPC90ZXh0PgogIDx0ZXh0IHg9IjQwMCiGeT0iNDgiIGZvbnQtZmFtaWx5PSJBcmllhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0iIzY2NiIgdGV4dC1hbmnob3I9Im1pZGRsZSI+RGF0YSBmbG93cyBsZW0IHRvIHJpZ2h0IHRCm91Z2ggc2VxdWVudGlhbCBjb21tYW5kczwdGV4dD4KCiAgPCetLSBQaXBlbGluZSBTdGFnZXMgLS0+CiAgPHJlY3QgeD0iMzAiIHk9IjciIiB3aWR0aD0iMTAwIIiBoZWlnaHQ9IjgwIIByeD0iMTAiIGZpbGw9InVybCgjZmV0Y2hHcmFkKSigZmlsdGVyPSJ1cmwoI2RxbFNoYWVrdykiIz4KICAgICA8dGV4dCB4PSI4MCiGeT0iMTA1IiBmb250LWZhbWlseT0ibW9ub3NwYWNLIIbmb250LXNpemU9IjEyIIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0id2hpGUiiHRleHQtYW5jaG9yPSJtaWRkbGUipmZldGn0PC90ZXh0PgogIDx0ZXh0IHg9IjgwIIiB5PSIxMjUiIGZvbnQtZmFtaWx5PSJBcmllhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCiGZmlsbD0icmdiYSgyNTUsMjU1LDI1NSwwLjklpIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5sb2dzPC90ZXh0PgogIDx0ZXh0IHg9IjgwIIiB5PSIxMCiGZmlsbD0icmdiYSgyNTUsMjU1LDI1NSwwLjklpIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5BbGwgcMvj3JkczwdGV4dD4KCiAgPHBhdGggZD0iTTExMCwxMTUgTDE1NSwxMTUiIHN0cm9rZT0iIzY0NzQ4YiIgc3Ryb2tlLXdpxZHRoPSIxIiBmaWxsPSJub25lIiBtYXJrZXItZW5kPSJ1cmwoI2RxbEFycm93KSIvPgoKICAgICA8cmVjdCB4PSIxNjUiIHk9IjciIiB3aWR0aD0iMTIwIIiBoZWlnaHQ9IjgwIIByeD0iMTAiIGZpbGw9InVybCgjZmlsdGVyR3JhZERRTCkiIGZpbHRlcj0idXjsKCNkcWxTaGFkb3cpIi8+CiAgPHRleHQgeD0iMjI1IiB5PSIxMDUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTIiIGZvbnQtd2VpZ2h0PSJib2xkIIiBmaWxsPSJ3aG10ZSIgdGV4dC1hbmnob3I9Im1pZGRsZSI+fCBmaWx0ZXI8L3RleHQ+CiAgPHRleHQgeD0iMjI1IiB5PSIxMjUiIGZvbnQtZmFtaWx5PSJBcmllhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCiGZmlsbD0icmdiYSgyNTUsMjU1LDI1NSwwLjklpIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5sb2dsZXZlbCA9PSAiRVJST1IiPC90ZXh0PgogIDx0ZXh0IHg9IjIyNSIgeT0iMTQ1IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9InJnYmEoMjU1LDI1NSwyNTUsMC45KSIgdGV4dC1hbmnob3I9Im1pZGRsZSI+UmVkdWNlIHZvd3M8L3RleHQ+CgogIDxwYXRoIGQ9Ik0yODUsMTE1IEwzMTAsMTE1IiBzdHJva2U9IiM2NDc0OGiIiHN0cm9rZS13aWR0aD0iMiIgZmlsbD0ibm9uZSIgbWFya2VylWVuZD0idXjsKCNkcWxBcnJvdykiIz4KCiAgPHJlY3QgeD0iMzIwIIiB5PSI3NSIgd21kdGg9IjEyiMCiGaGVpZ2h0PSI4MCiGcng9IjEwIIiBmaWxsPSJ1cmwoI2ZpZWxkc0dyYWQpIiBmaWx0ZXI9InVybCgjZHFsu2hhZG93KSIvPgogIDx0ZXh0IHg9IjM4MCiGeT0iMTA1IiBmb250LWZhbWlseT0ibW9ub3NwYWNL



```
eT0iMjQyIiBmb250LWZhbWlseT0iQXJpYwesIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTIiIGZpbGw9IiM2NDc0OGIiPuKGkjwvdGV4dD4KCiAgPHJlY3QgeD0iMzUwIiB5PSIyMTUiIHdpZHRoPSIxMjAiIGhlaWdodD0iNTUiIHJ4PSI2IiBmaWxsPSIjZDFmYWU1Ii8+CiAgPHRleHQgeD0iNDEwIiB5PSIyMzUiIGZvbnQtZmFtaWx5PSJBcmhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCigZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IiMwNDc4NTciIHRleHQtYW5jaG9yPSJtaWRkbGUipjUwSyByZWnvcmRzPC90ZXh0PgogIDx0ZXh0IHg9IjQxMCigeT0iMjU1IiBmb250LWZhbWlseT0iQXJpYwesIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiMwNjRlm2IiIHRleHQtYW5jaG9yPSJtaWRkbGUipjIgY29sdW1uczwvdGV4dD4KCiAgPHRleHQgeD0iNDg1IiB5PSIyNDIiIGZvbnQtZmFtaWx5PSJBcmhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMiIgZmlsbD0iIzY0NzQ4YiI+4oaSPC90ZXh0PgoKICA8cmVjdCB4PSI1MDAiIHk9IjIxNSIgd2lkdGg9IjEyMCiGaGVpZ2h0PSI1NSIgcn9IjYiIGZpbGw9IiNLZGU5ZmUiLz4KICA8dGV4dCB4PSI1NjAiIHk9IjIzNSIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0iIzZkMjhk0SIgdGV4dC1hbmNob3I9Im1pZGRsZSI+MTAwIHJlY29yZHM8L3RleHQ+CiAgPHRleHQgeD0iNTYwIiB5PSIyNTUiIGZvbnQtZmFtaWx5PSJBcmhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCigZmlsbD0iIzRjMWQ5NSIgdGV4dC1hbmNob3I9Im1pZGRsZSI+UGVyIHNLcnZpY2U8L3RleHQ+Cgo gIDx0ZXh0IHg9IjYzNSIgeT0iMjQyIiBmb250LWZhbWlseT0iQXJpYwesIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTIiIGZpbGw9IiM2NDc0OGIiPuKGkjwvdGV4dD4KCiAgPHJlY3QgeD0iNjUwIiB5PSIyMTUiIHdpZHRoPSIxMDUiIGHlaWdodD0iNTUiIHJ4PSI2IiBmaWxsPSIjZmVmM2M3Ii8+CiAgPHRleHQgeD0iNzAyIiB5PSIyMzUiIGZvbnQtZmFtaWx5PSJBcmhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCigZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IiM5MjQwMGUiIHRleHQtYW5jaG9yPSJtaWRkbGUipjEwIHJlY29yZHM8L3RleHQ+CiAgPHRleHQgeD0iNzAyIiB5PSIyNTUiIGZvbnQtZmFtaWx5PSJBcmhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCigZmlsbD0iIzc4MzUwZiIgdGV4dC1hbmNob3I9Im1pZGRsZSI+VG9wIDEwPC90ZXh0Pg08L3N2Zz4K)
```

```
```
```

```
fetch logs, from: now() - 1h          // Data source and time range
| filter                      // Filter records
| fields ,                   // Select fields
| sort desc                  // Order results
| limit 100                  // Limit output
```
```

### ### 🚨 Critical DQL Rules (DQL ≠ SQL)

<input checked="" type="checkbox"/> DQL Syntax	<input type="checkbox"/> SQL Syntax
--	-------------------------------------

```
| `filter status == "ERROR"` | `WHERE status = 'ERROR'` |
| `in(field, {"a", "b"})` | `field IN ('a', 'b')` |
| `summarize {count = count()}` | `SELECT COUNT(*)` |
| `isNull(field)` | `field IS NULL` |
```

```
```python
// Basic log query – explore recent logs
fetch logs, from: now() - 1h
| fields timestamp, content, loglevel, dt.entity.host
| limit 10
```

```

```
```
python
// Query with time range options
fetch logs, from: now() - 24h, to: now() - 12h
| summarize {count = count()}, by: {loglevel}
| sort count desc
```

## 2. Filtering Logs

### Comparison Operators

Operator	Description	Example
`==`	Equals	`loglevel == "ERROR"`
`!=`	Not equals	`status != "NONE"`
`>`, `>=`	Greater than	`count > 100`
`<`, `<=`	Less than	`duration < 1000`

### Logical Operators

Operator	Description	Example
`AND`	Both conditions	`loglevel == "ERROR" AND isNotNull(host)`
`OR`	Either condition	`loglevel == "ERROR" OR loglevel == "WARN"`
`NOT`	Negate	`NOT contains(content, "health")`

```
python
// Filter by log level
fetch logs, from: now() - 1h
| filter loglevel == "ERROR"
| fields timestamp, content, dt.entity.host
| limit 20
```

```
python
// Multiple conditions with AND/OR
fetch logs, from: now() - 1h
| filter (loglevel == "ERROR" OR loglevel == "WARN")
    AND isNotNull(dt.entity.host)
| summarize {count = count()}, by: {loglevel, dt.entity.host}
| sort count desc
| limit 15
```

```
python
// Filter using in() for multiple values

```

```

fetch logs, from: now() - 1h
| filter in(loglevel, {"ERROR", "WARN", "INFO"})
| summarize {count = count()}, by: {loglevel}
| sort count desc
```

## 3. String Matching Functions

Function	Description	Example
`contains(str, substr)`	Substring match	`contains(content, "error")`
`startsWith(str, prefix)`	Prefix match	`startsWith(content, "[ERROR]")`
`endsWith(str, suffix)`	Suffix match	`endsWith(content, "failed")`
`matchesPhrase(str, phrase)`	Word boundary match	
`matchesPhrase(content, "connection refused")`		
`matchesValue(str, pattern)`	Exact or wildcard match	
`matchesValue(host, "web-*")` |

```python
// Search for specific content patterns
fetch logs, from: now() - 1h
| filter contains(content, "Exception")
| fieldsAdd content_preview = substring(content, from: 0, to: 100)
| fields timestamp, content_preview, dt.entity.host
| limit 15
```

```python
// Use matchesPhrase for word-boundary matching
fetch logs, from: now() - 1h
| filter matchesPhrase(content, "connection")
| fieldsAdd content_preview = substring(content, from: 0, to: 120)
| summarize {count = count()}, by: {content_preview}
| sort count desc
| limit 10
```

```python
// Exclude patterns with NOT
fetch logs, from: now() - 1h
| filter loglevel == "ERROR"
| filter NOT matchesPhrase(content, "health")
| filter NOT matchesPhrase(content, "heartbeat")
| fieldsAdd content_preview = substring(content, from: 0, to: 100)
| summarize {count = count()}, by: {content_preview}
| sort count desc
| limit 10
```

```

```

```
## 4. Field Selection and Transformation

### Field Commands

| Command | Description |
|-----|-----|
| `fields` | Select specific fields only |
| `fieldsAdd` | Add computed fields |
| `fieldsRemove` | Remove fields |

```python
// Add computed fields
fetch logs, from: now() - 1h
| filter loglevel == "ERROR"
| fieldsAdd severity = if(contains(content, "critical"), "CRITICAL",
                           else: if(contains(content, "fatal"), "FATAL",
                           else: "ERROR"))
| fieldsAdd content_length = stringLength(content)
| fields timestamp, severity, content_length, dt.entity.host
| limit 20
```

```python
// String manipulation functions
fetch logs, from: now() - 1h
| filter isNotNull(k8s.namespace.name)
| fieldsAdd pod_short = substring(k8s.pod.name, from: 0, to: 30)
| fields timestamp, k8s.namespace.name, pod_short, loglevel
| limit 10
```

## 5. DPL (Dynatrace Pattern Language) Parsing

DPL extracts structured data from unstructured log content.

### DPL Matchers

Matcher	Description	Matches
`INT`	Integer	`42`, `-17`
`DOUBLE`	Decimal	`3.14`, `-0.5`
`IPADDR`	IP address	`192.168.1.1`, `::1`
`LD`	Line data (to delimiter)	Any text
`WORD`	Word characters	`hello`, `user123`
`SPACE`	Whitespace	spaces, tabs
`JSON`	JSON structure	`{"key": "value"}`

```

```

### Pattern Syntax

Syntax	Description
`MATCHER:fieldname`	Extract to named field
`MATCHER`	Match but don't extract
`MATCHER?`	Optional matcher
`literal`	Match exact string
`(opt1\|opt2)`	Match alternatives

```python
// Parse log level from content (e.g., "[ERROR] message")
fetch logs, from: now() - 1h
| filter startsWith(content, "[")
| parse content, "'[' LD:parsed_level ']'"
| filter isNotNull(parsed_level)
| summarize {count = count()}, by: {parsed_level}
| sort count desc
| limit 10
```

```python
// Parse HTTP-style logs
// Example: GET /api/users 200 45ms
fetch logs, from: now() - 1h
| filter contains(content, "GET") OR contains(content, "POST")
| parse content, "LD:method SPACE '/' LD:path SPACE INT:status_code"
| filter isNotNull(status_code)
| summarize {count = count()}, by: {method, status_code}
| sort count desc
| limit 15
```

```python
// Parse key-value pairs
// Example: "user=john action=login status=success"
fetch logs, from: now() - 1h
| parse content, "'user=' LD:username SPACE"
| filter isNotNull(username)
| summarize {count = count()}, by: {username}
| sort count desc
| limit 10
```

```python
// Parse IP addresses from logs
fetch logs, from: now() - 1h

```

```
| parse content, "IPADDR:client_ip"
| filter isNotNull(client_ip)
| summarize {count = count()}, by: {client_ip}
| sort count desc
| limit 15
```

```

## ## 6. Working with JSON Logs

Many applications emit JSON-formatted logs. DQL provides tools to work with JSON content.

```
```python
// Find JSON-formatted logs
fetch logs, from: now() - 1h
| filter startsWith(content, "{}")
| fieldsAdd content_preview = substring(content, from: 0, to: 150)
| summarize {count = count()}, by: {content_preview}
| sort count desc
| limit 10
```

```

```
```python
// Parse JSON and extract fields
fetch logs, from: now() - 1h
| filter startsWith(content, "{}")
| parse content, "JSON:json_data"
| filter isNotNull(json_data)
| fields timestamp, json_data
| limit 10
```

```

```
```python
// Access nested JSON fields
fetch logs, from: now() - 1h
| filter startsWith(content, "{}")
| parse content, "JSON:json_data"
| filter isNotNull(json_data)
| fieldsAdd error_type = json_data[errorType]
| fieldsAdd status_val = json_data[status]
| filter isNotNull(status_val)
| summarize {count = count()}, by: {status_val, error_type}
| sort count desc
| limit 15
```

```

## ## 7. Null Handling

DQL uses three-valued logic. `NULL` comparisons require special functions.

|                                      |           |
|--------------------------------------|-----------|
| ❌ Wrong                              | ✅ Correct |
| ----- -----                          |           |
| `field == null`   `isNull(field)`    |           |
| `field != null`   `isNotNull(field)` |           |

```
```python
// Check for null values
fetch logs, from: now() - 1h
| summarize {
    total = count(),
    with_host = countIf(isNotNull(dt.entity.host)),
    without_host = countIf(isNull(dt.entity.host)),
    with_namespace = countIf(isNotNull(k8s.namespace.name)),
    without_namespace = countIf(isNull(k8s.namespace.name))
}
```

```python
// Use coalesce for default values
fetch logs, from: now() - 1h
| fieldsAdd effective_level = coalesce(loglevel, status, "UNKNOWN")
| summarize {count = count()}, by: {effective_level}
| sort count desc
```

```

## ## 8. Advanced Parsing Examples

```
```python
// Parse exception patterns
fetch logs, from: now() - 24h
| filter contains(content, "Exception") OR contains(content, "Error")
| parse content, "LD:exception_type 'Exception'"
| filter isNotNull(exception_type)
| summarize {count = count()}, by: {exception_type}
| sort count desc
| limit 15
```

```python
// Parse with optional fields
// Matches: "error code=123" or "error code=123 message=failed"
fetch logs, from: now() - 1h
| filter contains(content, "error")
| parse content, "'error' (SPACE 'code=' INT:error_code)? (SPACE 'message=' LD:error_msg)?"
| filter isNotNull(error_code)
```

```

```
| summarize {count = count()}, by: {error_code}
| sort count desc
| limit 10
```

```python
// Parse alternative formats
// Matches: "user=john", "username=john"
fetch logs, from: now() - 1h
| parse content, "('user='|'username=') LD:user_value"
| filter isNotNull(user_value)
| summarize {count = count()}, by: {user_value}
| sort count desc
| limit 10
```

```

---

## ## 📈 Summary

In this notebook, you learned:

- ✓ \*\*DQL fundamentals\*\* – Query structure and syntax rules
- ✓ \*\*Filtering\*\* – Comparison operators, logical operators, `in()`
- ✓ \*\*String matching\*\* – `contains`, `matchesPhrase`, `startsWith`
- ✓ \*\*Field manipulation\*\* – `fieldsAdd`, computed fields
- ✓ \*\*DPL parsing\*\* – Matchers (INT, LD, IPADDR, JSON)
- ✓ \*\*JSON handling\*\* – Parsing and accessing nested fields
- ✓ \*\*Null handling\*\* – `isNull`, `isNotNull`, `coalesce`

---

## ## ➡ Next Steps

Continue to **OPLOGS-06: Topology & Entity Context** to learn about entity context and relationships.

---

## ## 📖 References

- [DQL Reference](<https://docs.dynatrace.com/docs/platform/grail/dynatrace-query-language>)
- [DQL Functions](<https://docs.dynatrace.com/docs/platform/grail/dynatrace-query-language/functions>)
- [Dynatrace Pattern Language](<https://docs.dynatrace.com/docs/discover-dynatrace/platform/grail/dynatrace-pattern-language>)

- [DPL Architect Tool](<https://docs.dynatrace.com/docs/discover-dynatrace/platform/grail/dynatrace-pattern-language/dpl-architect>)