

🔍 Querying & Parsing Logs

```
> **Series:** 0PL0GS | **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 OPLOGS-01 through OPLOGS-04
-  Basic understanding of regular expressions (helpful)

1. DQL Fundamentals

Query Structure

```
! [DQL Pipeline Model]
```

```
(data:image/svg+xml;base64,PHN2YyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvbWJAwMC9zdmciIHZpZXhCb3g9IjAgMCA4MDAgMzAwIj4KICA8ZGVmcz4KICAgIDxsaW5lYXJHcmFkaWVudCBpZD0iZmV0Y2hHcmFkIiB4MT0iMCUiIHkxPSIwJSIgeDI9IjEwMCUiIHkyPSIxMDAlIj4KICAgICAgPHN0b3Agb2Zmc2V0PSIwJSIgc3R5bGU9InN0b3AtY29sb3I6IzNiODJmNjtzdG9wLW9wYWNpdHk6MSIgLz4KICAgICAgPHN0b3Agb2Zmc2V0PSIxMDAlIiBzdHlsZT0ic3RvcC1jb2xvcjojMjU2M2Vi03N0b3Atb3BhY2l0eToxIiAvPgogICAgPC9saW5lYXJHcmFkaWVudD4KICAgIDxsaW5lYXJHcmFkaWVudCBpZD0iZmV0Y2hHcmFkIiB4MT0iMCUiIHg5PSIxMDAlIiB5Mj0iMTAwJSI+CjAgICAgIDxzdG9wIG9mZnNldD0iMCUiIHNoewxLPSJzdG9wLWNvbG9y0iNlZjQ0NDQ7c3RvcC1vcGFjaXR50jEiIC8+CjAgICAgIDxzdG9wIG9mZnNldD0iMTAwJSIgc3R5bGU9InN0b3AtY29sb3I6I2RjMjYyNjtzdG9wLW9wYWNpdHk6MSIgLz4KICAgIDwvbGluZWZyR3JhZGlbnQ+CjAgICA8bGluZWZyR3JhZGlbnQgaWQ9ImZpZwxc0dyYWQiIHgxPSIwJSIgeTE9IjAlIiB4Mj0iMTAwJSIgeTI9IjEwMCUi
```

[illegible]

IiBmb250LXNpemU9IjEyIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0id2hpdGUiIHRleHQtYW5jaG9yPSJtaWRkbGUiPnwgZmllbGRzPC90ZXh0PgogIDx0ZXh0IHg9IjM4MCIgeT0iMTI1IiBmb250LWZhbWlseT0iQXJpYWwsIHhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9InJnYmEoMjU1LDI1NSwyNTUsMC45KSIGdGV4dC1hbmNob3I9Im1pZGRsZSI+dGltZXN0YW1wLCBjb250ZW50PC90ZXh0PgogIDx0ZXh0IHg9IjM4MCIgeT0iMTQ1IiBmb250LWZhbWlseT0iQXJpYWwsIHhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9InJnYmEoMjU1LDI1NSwyNTUsMC45KSIGdGV4dC1hbmNob3I9Im1pZGRsZSI+U2VsZWNOIGNvbHVtbnM8L3RleHQ+CgogIDxwYXRoIGQ9Ik00NDAsMTE1IEw0NjUsMTE1IiBzdHJva2U9IiM2NDc0OGIiIHNOcm9rZS13aWR0aD0iMiIgZmlsbD0ibm9uZSIgbWfya2VyLWVuZD0idXJsKCNkcWxBcnJvdykiLz4KCIAGPHJlY3QgeD0iNDc1IiB5PSI3NSIGd2lkdGg9IjEzMCIgaGVpZ2h0PSI4MCIgcng9IjEwIiBmaWxsPSJ1cmwoI3N1bW1hcml6ZUdyYWQpIiBmaWx0ZXI9InVyBCgjZHF5U2hhZG93KSIVPgogIDx0ZXh0IHg9IjU0MCIgeT0iMTA1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEyIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0id2hpdGUiIHRleHQtYW5jaG9yPSJtaWRkbGUiPnwg3VtbWfyaXplPC90ZXh0PgogIDx0ZXh0IHg9IjU0MCIgeT0iMTI1IiBmb250LWZhbWlseT0iQXJpYWwsIHhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9InJnYmEoMjU1LDI1NSwyNTUsMC45KSIGdGV4dC1hbmNob3I9Im1pZGRsZSI+Y291bnQoKSwgYnk6e3N2Y308L3RleHQ+CIAgPHRleHQgeD0iNTQwIiB5PSIXNDUiIGZvbnQtZmFtaWx5PSJBcmllbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIXMCIgZmlsbD0icmdiYSgyNTUsMjU1LDI1NSwwLjE0IiB0ZXh0LWFuY2hvcj0ibWlkZGx1Ij5BZ2dyZWdhGU8L3RleHQ+CgogIDxwYXRoIGQ9Ik02MDUsMTE1IEw2MzAsMTE1IiBzdHJva2U9IiM2NDc0OGIiIHNOcm9rZS13aWR0aD0iMiIgZmlsbD0ibm9uZSIgbWfya2VyLWVuZD0idXJsKCNkcWxBcnJvdykiLz4KCIAGPHJlY3QgeD0iNjQwIiB5PSI3NSIGd2lkdGg9IjEzMCIgaGVpZ2h0PSI4MCIgcng9IjEwIiBmaWxsPSJ1cmwoI3NvcnRHcmFkKSIGZmlsdGVyPSJ1cmwoI2RxbFNOYWRvdykiLz4KICA8dGV4dCB4PSI3MDUiIHK9IjEwNSIGZm9udC1mYW1pbHk9Im1vb9zcGFjZSIgZm9udC1zaXplPSIXMiIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IndoaXRlIiB0ZXh0LWFuY2hvcj0ibWlkZGx1Ij58IHNVcnQgfCBsaW1pdDwvdGV4dD4KICA8dGV4dCB4PSI3MDUiIHK9IjEyNSIGZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEyIiBmaWxsPSJyZ2JhKDI1NSwyNTUsMjU1LDAuOSkiIHRleHQtYW5jaG9yPSJtaWRkbGUiPmNvdW50IGRlc2M8L3RleHQ+CIAgPHRleHQgeD0iNzA1IiB5PSIXNDUiIGZvbnQtZmFtaWx5PSJBcmllbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIXMCIgZmlsbD0icmdiYSgyNTUsMjU1LDI1NSwwLjE0IiB0ZXh0LWFuY2hvcj0ibWlkZGx1Ij5PcmRlcicArIGxpbWl0PC90ZXh0PgoKICA8IS0tIERhdGEgRmxvdyBwaXN1YWxpemF0aW9uIC0tPgogIDxyZWNOIHg9IjMwIiB5PSIXNzUiIHdpZHRoPSI3NDAAiIGhlaWdodD0iMTEwIiBieD0iMTAiIGZpbGw9IiNmZmYiIHNOcm9rZT0iI2UyZThmMCIgc3Ryb2tLLXdpZHRoPSIyIi8+CIAgPHRleHQgeD0iNDAwIiB5PSIyMDAiIGZvbnQtZmFtaWx5PSJBcmllbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIXMiIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9ImZmZmIiHRleHQtYW5jaG9yPSJtaWRkbGUiPkrRhdGEgVHJhbnNmb3JtYXRpb24gYXQgRWFjaCBTdGFnZWvdGV4dD4KCIAGPCEtLSBtdGFnZSBkZXhwaWxzIC0tPgogIDxyZWNOIHg9IjUwIiB5PSIyMTUiIHdpZHRoPSIXMjAAiIGhlaWdodD0iNTUuIiHJ4PSI2IiBmaWxsPSIjZGJlYWZlIi8+CIAgPHRleHQgeD0iMTEwIiB5PSIyMzUiIGZvbnQtZmFtaWx5PSJBcmllbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIXMCIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9Imx0ZTQwYWYiIHRleHQtYW5jaG9yPSJtaWRkbGUiPjFNIHJlY29yZHM8L3RleHQ+CIAgPHRleHQgeD0iMTEwIiB5PSIyNTUuIIGZvbnQtZmFtaWx5PSJBcmllbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIXMCIgZmlsbD0iIzF1M2E4YSIGdGV4dC1hbmNob3I9Im1pZGRsZSI+QWxsIGZpZWxkc2wvdGV4dD4KCIAGPHRleHQgeD0iMTg1IiB5PSIyNDUiIGZvbnQtZmFtaWx5PSJBcmllbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIXMiIgZmlsbD0iIzY0NzQ4YiI+4oaSPC90ZXh0PgoKICA8cmVjdCB4PSIyMDAAiIHK9IjIxNSIGd2lkdGg9IjEwIiBmaWxsPSJyZ2h0PSI1NSIGcng9IjEwIiIGZpbGw9IiNmZWYyZjEiLz4KICA8dGV4dCB4PSIyNjAAiIHK9IjIzNSIGZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0iI2RjMjYyNiIgZm9udC1hbmNob3I9Im1pZGRsZSI+NTBLIHJlY29yZHM8L3RleHQ+CIAgPHRleHQgeD0iMjYwIiB5PSIyNTUuIIGZvbnQtZmFtaWx5PSJBcmllbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIXMCIgZmlsbD0iIzdmMWwqZCIgdGV4dC1hbmNob3I9Im1pZGRsZSI+RXJyb3JzIG9ubHk8L3RleHQ+CgogIDx0ZXh0IHg9IjMzNSIG


```
```
```

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

```

```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 ****OPLGS-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>)