```
# Your First Queries

> **Series:** ONBRD | **Notebook:** 8 of 10 | **Created:** December 2025

## Learning DQL Fundamentals

Dynatrace Query Language (DQL) is how you access data in Grail. This notebook
introduces the core concepts and patterns you'll use daily.

---

## Table of Contents

1. DQL Basics
2. The Pipeline Model
3. Fetching Data
4. Filtering
5. Selecting Fields
6. Aggregating with Summarize
7. Sorting and Limiting
8. Time Ranges
9. Common Patterns
10. Next Steps

---

## Prerequisites

- Dynatrace environment with data (ONBRD-05, ONBRD-06, ONBRD-07)
- DQL query permissions
- Access to Notebooks or the DQL query interface

## 1. DQL Basics

DQL is a **pipeline-based query language**—not SQL. Data flows through a
series of commands connected by the pipe (`|`) operator.

![DQL Pipeline]
```
(data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdm
ciIHZpZXdCb3g9IjAgMCA4MDAgMjAwIj4KICA8ZGVmcz4KICAgIDxsaW5lYXJHcmFkaWVudCBpZD0
iZmV0Y2hHcmFkIiB4MT0iMCUiIHkxPSIwJSIgeDI9IjEwMCUiIHkyPSIxMDAlIj4KICAgICAgPHN0
b3Agb2Zmc2V0PSIwJSIgc3R5bGU9InN0b3AtY29sb3I6IzE0OTZmZjtzdG9wLW9wYWNpdHk6MSIgL
z4KICAgICAgPHN0b3Agb2Zmc2V0PSIxMDAlIiBzdHlsZT0ic3RvcC1jb2xvcjojMGE2NGJjO3N0b3
Atb3BhY2l0eToxIiAvPgogICAgPC9saW5lYXJHcmFkaWVudD4KICAgIDxsaW5lYXJHcmFkaWVudCB
pZD0iZmlsdGVyR3JhZCIgeDE9IjAlIiB5MT0iMCUiIHgyPSIxMDAlIiB5Mj0iMTAwJSI+CiAgICAg
IDxzdG9wIG9mZnNldD0iMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiMxMGI5ODE7c3RvcC1vcGFjaXR5O
jEiIC8+CiAgICAgIDxzdG9wIG9mZnNldD0iMTAwJSIgc3R5bGU9InN0b3AtY29sb3I6IzA1OTY2OT

```
tzdG9wLW9wYWNpdHk6MSIgLz4KICAgIDwvbGluZWFyR3JhZGllbnQ+CiAgICA8bGluZWFyR3JhZGl
lbnQgaWQ9ImZpZWxkc0dyYWQiIHgxPSIwJSIgeTE9IjAlIiB4Mj0iMTAwJSIgeTI9IjEwMCUiPgog
ICAgICA8c3RvcCBvZmZzZXQ9IjAlIiBzdHlsZT0ic3RvcC1jb2xvcjojZjU5ZTBiO3N0b3Atb3BhY
2l0eToxIiAvPgogICAgICA8c3RvcCBvZmZzZXQ9IjEwMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiNkOT
c3MDY7c3RvcC1vcGFjaXR5OjEiIC8+CiAgICA8L2xpbmVhckdyYWRpZW50PgogICAgPGxpbmVhckd
yYWRpZW50IGlkPSJzb3J0R3JhZCIgeDE9IjAlIiB5MT0iMCUiIHgyPSIxMDAlIiB5Mj0iMTAwJSI+
CiAgICAgIDxzdG9wIG9mZnNldD0iMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiM4YjVjZjY7c3RvcC1vc
GFjaXR5OjEiIC8+CiAgICAgIDxzdG9wIG9mZnNldD0iMTAwJSIgc3R5bGU9InN0b3AtY29sb3I6Iz
djM2FlZDtzdG9wLW9wYWNpdHk6MSIgLz4KICAgIDwvbGluZWFyR3JhZGllbnQ+CiAgICA8ZmlsdGV
yIGlkPSJwaXBlbGluZVNoYWRvdyI+CiAgICAgIDxmZURyb3BTaGFkb3cgZHg9IjEiIGR5PSIxIiBz
dGREZXZpYXRpb249IjIiIGZsb29kLW9wYWNpdHk9IjAuMTUiLz4KICAgIDwvZmlsdGVyPgogICAgP
G1hcmtlciBpZD0icGlwZUFycm93IiBtYXJrZXJXaWR0aD0iMTAiIG1hcmtlckhlaWdodD0iNyIgcm
VmWD0iOSIgcmVmWT0iMy41IiBvcmllbnQ9ImF1dG8iPgogICAgICA8cG9seWdvbiBwb2ludHM9IjA
gMCwgMTAgMy41LCAwIDciIGZpbGw9IiM2NDc0OGIiLz4KICAgIDwvbWFya2VyPgogIDwvZGVmcz4K
CiAgPCEtLSBCYWNrZ3JvdW5kIC0tPgogIDxyZWN0IHdpZHRoPSI4MDAiIGhlaWdodD0iMjAwIiBma
WxsPSIjZjhmOWZhIiByeD0iMTAiLz4KCiAgPCEtLSBUaXRsZSAtLT4KICA8dGV4dCB4PSI0MDAiIH
k9IjI4IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTgiIGZvbnQ
td2VpZ2h0PSJib2xkIiBmaWxsPSIjMzMzIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5EUUwgUGlwZWxp
bmUgTW9kZWw8L3RleHQ+CiAgPHRleHQgeD0iNDAwIiB5PSI0OCIgZm9udC1mYW1pbHk9IkFyaWFsL
CBzYW5zLXNlcmlmIiBmb250LXNpemU9IjExIiBmaWxsPSIjNjY2IiB0ZXh0LWFuY2hvcj0ibWlkZG
xlIj5EYXRhIGZsb3dzIHRocm91Z2ggY29tbWFuZHMgY29ubmVjdGVkIGJ5IHRoZSBwaXBlICh8KSB
vcGVyYXRvcjwvdGV4dD4KCiAgPCEtLSBQaXBlbGluZSBTdGFnZXMgLS0+CiAgPCEtLSBGZXRjaCAt
LT4KICA8cmVjdCB4PSI0MCIgeT0iNzUiIHdpZHRoPSIxNDAiIGhlaWdodD0iNzAiIHJ4PSIxMCIgZ
mlsbD0idXJsKCNmZXRjaEdyYWQpIiBmaWx0ZXI9InVybCgjcGlwZWxpbmVTaGFkb3cpIi8+CiAgPH
RleHQgeD0iMTEwIiB5PSIxMDUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1
zaXplPSIxMyIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IndoaXRlIiB0ZXh0LWFuY2hvcj0ibWlk
ZGxlIj5mZXRjaDwvdGV4dD4KICA8dGV4dCB4PSIxMTAiIHk9IjEyNSIgZm9udC1mYW1pbHk9IkFya
WFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjExIiBmaWxsPSJyZ2JhKDI1NSwyNTUsMjU1LDAuOS
kiIHRleHQtYW5jaG9yPSJtaWRkbGUiPmxvZ3M8L3RleHQ+CgogIDwhLS0gQXJyb3cgLS0+CiAgPHB
hdGggZD0iTTE4NSwxMTAgTDIxNSwxMTAiIHN0cm9rZT0iIzY0NzQ4YiIgc3Ryb2tlLXdpZHRoPSIy
IiBmaWxsPSJub25lIiBtYXJrZXItZW5kPSJ1cmwoI3BpcGVBcnJvdykiLz4KCiAgPCEtLSBGaWx0Z
XIgLS0+CiAgPHJlY3QgeD0iMjIwIiB5PSI3NSIgd2lkdGg9IjE0MCIgaGVpZ2h0PSI3MCIgcng9Ij
EwIiBmaWxsPSJ1cmwoI2ZpbHRlckdyYWQpIiBmaWx0ZXI9InVybCgjcGlwZWxpbmVTaGFkb3cpIi8
+CiAgPHRleHQgeD0iMjkwIiB5PSIxMDUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIg
Zm9udC1zaXplPSIxMyIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IndoaXRlIiB0ZXh0LWFuY2hvc
j0ibWlkZGxlIj5maWx0ZXI8L3RleHQ+CiAgPHRleHQgeD0iMjkwIiB5PSIxMjUiIGZvbnQtZmFtaW
x5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0icmdiYSgyNTUsMjU1LDI
1NSwwLjkpIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5zdGF0dXM8L3RleHQ+CgogIDwhLS0gQXJyb3cg
LS0+CiAgPHBhdGggZD0iTTM2NSwxMTAgTDM5NSwxMTAiIHN0cm9rZT0iIzY0NzQ4YiIgc3Ryb2tlL
XdpZHRoPSIyIiBmaWxsPSJub25lIiBtYXJrZXItZW5kPSJ1cmwoI3BpcGVBcnJvdykiLz4KCiAgPC
EtLSBGaWVsZHMgLS0+CiAgPHJlY3QgeD0iNDAwIiB5PSI3NSIgd2lkdGg9IjE0MCIgaGVpZ2h0PSI
3MCIgcng9IjEwIiBmaWxsPSJ1cmwoI2ZpZWxkc0dyYWQpIiBmaWx0ZXI9InVybCgjcGlwZWxpbmVT
aGFkb3cpIi8+CiAgPHRleHQgeD0iNDcwIiB5PSIxMDUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fuc
y1zZXJpZiIgZm9udC1zaXplPSIxMyIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IndoaXRlIiB0ZX
h0LWFuY2hvcj0ibWlkZGxlIj5maWVsZHM8L3RleHQ+CiAgPHRleHQgeD0iNDcwIiB5PSIxMjUiIGZ
vbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0icmdiYSgy
NTUsMjU1LDI1NSwwLjkpIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5zZWxlY3Q8L3RleHQ+CgogIDwhL
S0gQXJyb3cgLS0+CiAgPHBhdGggZD0iTTU0NSwxMTAgTDU3NSwxMTAiIHN0cm9rZT0iIzY0NzQ4Yi
```

Igc3Ryb2tlLXdpZHRoPSIyIiBmaWxsPSJub25lIiBtYXJrZXItZW5kPSJ1cmwoI3BpcGVBcnJvdyk
iLz4KCiAgPCEtLSBTb3J0IC0tPgogIDxyZWN0IHg9IjU4MCIgeT0iNzUiIHdpZHRoPSIxNDAiIGhl
aWdodD0iNzAiIHJ4PSIxMCIgZmlsbD0idXJsKCNzb3J0R3JhZCkiIGZpbHRlcj0idXJsKCNwaXBlb
GluZVNoYWRvdykiLz4KICA8dGV4dCB4PSI2NTAiIHk9IjEwNSIgZm9udC1mYW1pbHk9IkFyaWFsLC
BzYW5zLXNlcmlmIiBmb250LXNpemU9IjEzIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0id2hpdGU
iIHRleHQtYW5jaG9yPSJtaWRkbGUiPnNvcnQ8L3RleHQ+CiAgPHRleHQgeD0iNjUwIiB5PSIxMjUi
IGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0icmdiY
SgyNTUsMjU1LDI1NSwwLjkpIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5vcmRlcjwvdGV4dD4KCiAgPC
EtLSBGbG93IEluZGljYXRvcnMgLS0+CiAgPHRleHQgeD0iMTEwIiB5PSIxNjUiIGZvbnQtZmFtaWx
5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzY0NzQ4YiIgdGV4dC1h
bmNob3I9Im1pZGRsZSI+QWxsIGxvZ3M8L3RleHQ+CiAgPHRleHQgeD0iMjkwIiB5PSIxNjUiIGZvb
nQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzY0NzQ4Yi
IgdGV4dC1hbmNob3I9Im1pZGRsZSI+T25seSBlcnJvcnM8L3RleHQ+CiAgPHRleHQgeD0iNDcwIiB
5PSIxNjUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmls
bD0iIzY0NzQ4YiIgdGV4dC1hbmNob3I9Im1pZGRsZSI+SnVzdCBmaWVsZHM8L3RleHQ+CiAgPHRle
HQgeD0iNDcwIiB5PSIxNzgiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaX
plPSIxMCIgZmlsbD0iIzY0NzQ4YiIgdGV4dC1hbmNob3I9Im1pZGRsZSI+d2UgbmVlZDwvdGV4dD4
KICA8dGV4dCB4PSI2NTAiIHk9IjE2NSIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBm
b250LXNpemU9IjEwIiBmaWxsPSIjNjQ3NDhiIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5PcmRlcmVkP
C90ZXh0PgogIDx0ZXh0IHg9IjY1MCIgeT0iMTc4IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2
VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM2NDc0OGIiIHRleHQtYW5jaG9yPSJtaWRkbGUiPm9
1dHB1dDwvdGV4dD4KCiAgPCEtLSBBcnJvdyBpbmRpY2F0b3JzIGJlbG93IGJveGVzIC0tPgogIDxw
YXRoIGQ9Ik0xMTAsMTQ1IEwxMTAsMTU1IiBzdHJva2U9IiM2NDc0OGIiIHN0cm9rZS13aWR0aD0iM
SIgZmlsbD0ibm9uZSIvPgogIDxwYXRoIGQ9Ik0yOTAsMTQ1IEwyOTAsMTU1IiBzdHJva2U9IiM2ND
c0OGIiIHN0cm9rZS13aWR0aD0iMSIgZmlsbD0ibm9uZSIvPgogIDxwYXRoIGQ9Ik00NzAsMTQ1IEw
0NzAsMTU1IiBzdHJva2U9IiM2NDc0OGIiIHN0cm9rZS13aWR0aD0iMSIgZmlsbD0ibm9uZSIvPgog
IDxwYXRoIGQ9Ik02NTAsMTQ1IEw2NTAsMTU1IiBzdHJva2U9IiM2NDc0OGIiIHN0cm9rZS13aWR0a
D0iMSIgZmlsbD0ibm9uZSIvPgo8L3N2Zz4K)

~~~
### DQL vs SQL

| DQL | SQL | Note |
|-----|-----|------|
| `fetch logs` | `SELECT * FROM logs` | Start with data source |
| `\| filter x == "y"` | `WHERE x = 'y'` | Use `==`, double quotes |
| `\| fields a, b` | `SELECT a, b` | Field selection after fetch |
| `\| summarize count()` | `SELECT COUNT(*)` | Aggregation command |
| `by: {field}` | `GROUP BY field` | Grouping syntax |
| `{"a", "b"}` | `('a', 'b')` | Array syntax (curly braces) |

## 2. The Pipeline Model

Each command in the pipeline operates on the output of the previous command:

```dql
fetch logs                        // 1. Get all logs
| filter loglevel == "error"     // 2. Keep only errors
~~~

~~~
| filter timestamp > now() - 1h  // 3. Last hour only
| fields timestamp, content      // 4. Select columns
| sort timestamp desc            // 5. Order by time
| limit 100                      // 6. Take first 100
```

### Order Matters

- **Filter early** - Reduces data before expensive operations
- **Select fields** - Reduces memory usage
- **Aggregate** - Summarize before sorting
- **Sort** - Order the final results
- **Limit** - Control output size

## 3. Fetching Data

Every DQL query starts with a `fetch` command specifying the data source.

```dql
// Fetch logs (most common)
fetch logs
| limit 10
```

```dql
// Fetch spans (distributed traces)
fetch spans
| limit 10
```

```dql
// Fetch entity data (hosts)
fetch dt.entity.host
| limit 10
```

```dql
// Fetch problems
fetch dt.davis.problems
| limit 10
```

### Common Data Sources

| Source | Description |
|--------|-------------|
| `logs` | Log records |
| `spans` | Distributed trace spans |
~~~

~~~
| `events` | System events |
| `bizevents` | Business events |
| `dt.entity.host` | Host entities |
| `dt.entity.service` | Service entities |
| `dt.entity.process_group` | Process group entities |
| `dt.davis.problems` | Detected problems |

## 4. Filtering

Use `filter` to narrow results. Filter as early as possible for performance.

```dql
// Filter by equality
fetch logs
| filter loglevel == "error"
| limit 20
```

```dql
// Filter with multiple conditions (AND)
fetch logs
| filter loglevel == "error"
| filter timestamp > now() - 1h
| limit 20
```

```dql
// Filter with OR condition
fetch logs
| filter loglevel == "error" or loglevel == "warn"
| limit 20
```

```dql
// Filter using IN for multiple values
fetch logs
| filter in(loglevel, {"error", "warn", "fatal"})
| limit 20
```

```dql
// Filter with string matching
fetch logs
| filter contains(content, "timeout")
| limit 20
```

### Filter Operators
~~~

~~~
| Operator | Example | Description |
|----------|---------|-------------|
| `==` | `field == "value"` | Equals |
| `!=` | `field != "value"` | Not equals |
| `>`, `<` | `count > 10` | Greater/less than |
| `>=`, `<=` | `count >= 10` | Greater/less or equal |
| `and`, `or` | `a == 1 and b == 2` | Logical operators |
| `in()` | `in(field, {"a", "b"})` | Value in set |
| `contains()` | `contains(field, "text")` | Substring match |
| `isNull()` | `isNull(field)` | Field is null |
| `isNotNull()` | `isNotNull(field)` | Field is not null |

## 5. Selecting Fields

Use `fields` to select specific columns. This improves readability and
performance.

```dql
// Select specific fields from logs
fetch logs
| fields timestamp, loglevel, log.source, content
| limit 20
```

```dql
// Create calculated fields
fetch spans
| fields span.name,
         duration,
         duration_ms = duration / 1000000.0
| limit 20
```

```dql
// Rename fields with aliases
fetch dt.entity.host
| fields name = entity.name,
         status = state,
         os = osType
| limit 20
```

### fieldsAdd vs fields

| Command | Effect |
|---------|--------|
| `fields` | Keeps only specified fields |
~~~

~~~
| `fieldsAdd` | Adds new fields, keeps all existing |

```dql
// fieldsAdd keeps existing fields and adds new ones
fetch spans
| fieldsAdd duration_ms = duration / 1000000.0
| fields span.name, duration, duration_ms
| limit 10
```

## 6. Aggregating with Summarize

Use `summarize` to aggregate data. Combine with `by:` for grouping.

```dql
// Simple count
fetch logs, from: now() - 1h
| summarize total_logs = count()
```

```dql
// Count by group
fetch logs, from: now() - 1h
| summarize log_count = count(), by: {loglevel}
| sort log_count desc
```

```dql
// Multiple aggregations
fetch spans, from: now() - 1h
| filter span.kind == "server"
| summarize
    request_count = count(),
    avg_duration = avg(duration),
    max_duration = max(duration),
    by: {service.name}
| sort request_count desc
| limit 20
```

```dql
// Conditional counting
fetch spans, from: now() - 1h
| filter span.kind == "server"
| summarize
    total = count(),
    errors = countIf(span.status_code == "error"),
    by: {service.name}
~~~

~~~~
| fieldsAdd error_rate = 100.0 * errors / total
| sort error_rate desc
| limit 20
```

### Common Aggregation Functions

| Function | Description |
|----------|-------------|
| `count()` | Count records |
| `countIf(condition)` | Count where condition is true |
| `sum(field)` | Sum values |
| `avg(field)` | Average value |
| `min(field)` | Minimum value |
| `max(field)` | Maximum value |
| `percentile(field, 95)` | 95th percentile |

## 7. Sorting and Limiting

Control output order and size.

```dql
// Sort descending (newest first)
fetch logs, from: now() - 1h
| fields timestamp, loglevel, content
| sort timestamp desc
| limit 20
```

```dql
// Sort by multiple fields
fetch spans, from: now() - 1h
| filter span.kind == "server"
| summarize request_count = count(), by: {service.name}
| sort request_count desc
| limit 10
```

```dql
// Find slowest spans
fetch spans, from: now() - 1h
| fields span.name, service.name, duration
| sort duration desc
| limit 10
```

## 8. Time Ranges
~~~~

~~~
Control the time range for your queries.

```dql
// Last hour (using from: parameter)
fetch logs, from: now() - 1h
| summarize count()
```

```dql
// Specific time range
fetch logs, from: now() - 24h, to: now() - 12h
| summarize count()
```

```dql
// Last 7 days
fetch dt.davis.problems, from: now() - 7d
| summarize problem_count = count(), by: {status}
```

### Time Units

| Unit | Example | Description |
|------|---------|-------------|
| `s` | `now() - 30s` | Seconds |
| `m` | `now() - 15m` | Minutes |
| `h` | `now() - 2h` | Hours |
| `d` | `now() - 7d` | Days |

## 9. Common Patterns

Here are patterns you'll use frequently.

### Error Investigation

```dql
// Find error logs with context
fetch logs, from: now() - 1h
| filter loglevel == "error"
| fields timestamp, log.source, content
| sort timestamp desc
| limit 50
```

### Service Performance

```dql
// Service response time summary
~~~

~~~
fetch spans, from: now() - 1h
| filter span.kind == "server"
| summarize
    requests = count(),
    avg_ms = avg(duration) / 1000000.0,
    p95_ms = percentile(duration, 95) / 1000000.0,
    by: {service.name}
| sort requests desc
| limit 20
```

### Log Volume Analysis

```dql
// Log volume by source and severity
fetch logs, from: now() - 1h
| summarize log_count = count(), by: {log.source, loglevel}
| sort log_count desc
| limit 30
```

### Entity Inventory

```dql
// Host inventory with details
fetch dt.entity.host
| fields
    name = entity.name,
    state,
    os = osType,
    cores = cpuCores
| sort name
| limit 50
```

## 10. Next Steps

With DQL fundamentals covered:

1. **ONBRD-09: Setting Up Alerts** - Configure alerting and notifications
2. Practice with your own data
3. Explore the DQL documentation for advanced functions
4. Try creating queries in Notebooks

### DQL Checklist

- [ ] Understand the pipeline model
- [ ] Can fetch different data types
~~~

```
- [ ] Can filter with conditions
- [ ] Can select and calculate fields
- [ ] Can aggregate with summarize
- [ ] Can sort and limit results
- [ ] Can specify time ranges

---

## Summary

In this notebook, you learned:

- DQL uses a pipeline model (not SQL)
- `fetch` starts every query
- `filter` narrows results
- `fields` selects and calculates columns
- `summarize` aggregates data
- `sort` and `limit` control output
- Time ranges with `from:` and `to:`

---

## 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)
- [DQL Commands](https://docs.dynatrace.com/docs/platform/grail/dynatrace-query-language/commands)
- [DQL Examples](https://docs.dynatrace.com/docs/platform/grail/dynatrace-query-language/dql-examples)
```