🔎 Querying Spans with DQL

> **Series:** SPANS | **Notebook:** 2 of 8 | **Created:** December 2025

Mastering Span Queries in Dynatrace

This notebook covers essential techniques for querying and filtering span data to find exactly what you need. You'll learn to filter by service, operation, and attributes to quickly locate relevant traces.

Table of Contents

Prerequisites

Before starting this notebook, ensure you have:

- ✅ Completed **SPANS-01: Fundamentals**
- ✅ Access to a Dynatrace environment with span data
- ✅ DQL query permissions

1. DQL is NOT SQL!

⚠️ **CRITICAL:** DQL has different syntax from SQL. Memorize these differences:

("

IDxzdG9wIG9mZnNldD0iMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiNmOTczMTY7c3RvcC1vcGFjaXR5O
jEiIC8+CiAgICAgIDxzdG9wIG9mZnNldD0iMTAwJSIgc3R5bGU9InN0b3AtY29sb3I6I2VhNTgwYz
tzdG9wLW9wYWNpdHk6MSIgLz4KICAgIDwvbGluZWFyR3JhZGllbnQ+CiAgICA8bGluZWFyR3JhZGl
lbnQgaWQ9InRyYW5zZm9ybUdyYWQiIHgxPSIwJSIgeTE9IjAlIiB4Mj0iMTAwJSIgeTI9IjEwMCUi
PgogICAgICA8c3RvcCBvZmZzZXQ9IjAlIiBzdHlsZT0ic3RvcC1jb2xvcjojMTQ5NmZmO3N0b3Atb
3BhY2l0eToxIiAvPgogICAgICA8c3RvcCBvZmZzZXQ9IjEwMCUiIHN0eWxlPSJzdG9wLWNvbG9yOi
MwYTY0YmM7c3RvcC1vcGFjaXR5OjEiIC8+CiAgICA8L2xpbmVhckdyYWRpZW50PgogICAgPGxpbmV
hckdyYWRpZW50IGlkPSJvdXRwdXRHcmFkIiB4MT0iMCUiIHkxPSIwJSIgeDI9IjEwMCUiIHkyPSIx
MDAlIj4KICAgICAgPHN0b3Agb2Zmc2V0PSIwJSIgc3R5bGU9InN0b3AtY29sb3I6IzIyYzU1ZTtzd
G9wLW9wYWNpdHk6MSIgLz4KICAgICAgPHN0b3Agb2Zmc2V0PSIxMDAlIiBzdHlsZT0ic3RvcC1jb2
xvcjojMTZhMzRhO3N0b3Atb3BhY2l0eToxIiAvPgogICAgPC9saW5lYXJHcmFkaWVudD4KICAgIDx
maWx0ZXIgaWQ9InBpcGVsaW5lU2hhZG93Ij4KICAgICAgPGZlRHJvcFNoYWRvdyBkeD0iMiIgZHk9
IjIiIHN0ZERldmlhdGlvbj0iMyIgZmxvb2Qtb3BhY2l0eT0iMC4xNSIvPgogICAgPC9maWx0ZXI+C
iAgICA8bWFya2VyIGlkPSJwaXBlQXJyb3ciIG1hcmtlcldpZHRoPSIxMCIgbWFya2VySGVpZ2h0PS
I3IiByZWZYPSI5IiByZWZZPSIzLjUiIG9yaWVudD0iYXV0byI+CiAgICAgIDxwb2x5Z29uIHBvaW5
0cz0iMCAwLCAxMCAzLjUsIDAgNyIgZmlsbD0iIzk0YTNiOCIvPgogICAgPC9tYXJrZXI+CiAgPC9k
ZWZzPgoKICA8IS0tIEJhY2tncm91bmQgLS0+CiAgPHJlY3Qgd2lkdGg9IjgwMCIgaGVpZ2h0PSIyO
DAiIGZpbGw9IiNmOGY5ZmEiIHJ4PSIxMCIvPgoKICA8IS0tIFRpdGxlIC0tPgogIDx0ZXh0IHg9Ij
QwMCIgeT0iMjgiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxOCI
gZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IiMzMzMiIHRleHQtYW5jaG9yPSJtaWRkbGUiPkRRTCBQ
aXBlbGluZSBNb2RlbCBmb3IgU3BhbnM8L3RleHQ+CgogIDwhLS0gRmV0Y2ggc3RhZ2UgLS0+CiAgP
HJlY3QgeD0iNDAiIHk9IjYwIiB3aWR0aD0iMTQwIiBoZWlnaHQ9IjgwIiByeD0iOCIgZmlsbD0idX
JsKCNmZXRjaEdyYWQpIiBmaWx0ZXI9InVybCgjcGlwZWxpbmVTaGFkb3cpIi8+CiAgPHRleHQgeD0
iMTEwIiB5PSI5NSIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEz
IiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0id2hpdGUiIHRleHQtYW5jaG9yPSJtaWRkbGUiPmZld
GNoIHNwYW5zPC90ZXh0PgogIDx0ZXh0IHg9IjExMCIgeT0iMTE1IiBmb250LWZhbWlseT0iQXJpYW
wsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9InJnYmEoMjU1LDI1NSwyNTUsMC45KSI
gdGV4dC1hbmNob3I9Im1pZGRsZSI+R2V0IGZyb20gR3JhaWw8L3RleHQ+CgogIDwhLS0gQXJyb3cg
MSAtLT4KICA8bGluZSB4MT0iMTgwIiB5MT0iMTAwIiB4Mj0iMjE1IiB5Mj0iMTAwIiBzdHJva2U9I
iM5NGEzYjgiIHN0cm9rZS13aWR0aD0iMyIgbWFya2VyLWVuZD0idXJsKCNwaXBlQXJyb3cpIi8+Ci
AgPHRleHQgeD0iMTk3IiB5PSI5MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb25
0LXNpemU9IjExIiBmaWxsPSIjNjQ3NDhiIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj58PC90ZXh0PgoK
ICA8IS0tIEZpbHRlciBzdGFnZSAtLT4KICA8cmVjdCB4PSIyMjAiIHk9IjYwIiB3aWR0aD0iMTQwI
iBoZWlnaHQ9IjgwIiByeD0iOCIgZmlsbD0idXJsKCNmaWx0ZXJHcmFkKSIgZmlsdGVyPSJ1cmwoI3
BpcGVsaW5lU2hhZG93KSIvPgogIDx0ZXh0IHg9IjI5MCIgeT0iOTUiIGZvbnQtZmFtaWx5PSJBcml
hbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMyIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9Indo
aXRlIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5maWx0ZXI8L3RleHQ+CiAgPHRleHQgeD0iMjkwIiB5P
SIxMTUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD
0icmdiYSgyNTUsMjU1LDI1NSwwLjkpIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5SZWR1Y2UgZGF0YSB
lYXJseTwvdGV4dD4KCiAgPCEtLSBBcnJvdyAyIC0tPgogIDxsaW5lIHgxPSIzNjAiIHkxPSIxMDAi
IHgyPSIzOTUiIHkyPSIxMDAiIHN0cm9rZT0iIzk0YTNiOCIgc3Ryb2tlLXdpZHRoPSIzIiBtYXJrZ
XItZW5kPSJ1cmwoI3BpcGVBcnJvdykiLz4KICA8dGV4dCB4PSIzNzciIHk9IjkwIiBmb250LWZhbW
lseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTEiIGZpbGw9IiM2NDc0OGIiIHRleHQ
tYW5jaG9yPSJtaWRkbGUiPnw8L3RleHQ+CgogIDwhLS0gVHJhbnNmb3JtIHN0YWdlIC0tPgogIDxy
ZWN0IHg9IjQwMCIgeT0iNjAiIHdpZHRoPSIxNDAiIGhlaWdodD0iODAiIHJ4PSI4IiBmaWxsPSJ1c
mwoI3RyYW5zZm9ybUdyYWQpIiBmaWx0ZXI9InVybCgjcGlwZWxpbmVTaGFkb3cpIi8+CiAgPHRleH
QgeD0iNDcwIiB5PSI5MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU
9IjEzIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0id2hpdGUiIHRleHQtYW5jaG9yPSJtaWRkbGUi

PmZpZWxkcyAvIHN1bW1hcml6ZTwvdGV4dD4KICA8dGV4dCB4PSI0NzAiIHk9IjExMCIgZm9udC1mY
W1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSJyZ2JhKDI1NSwyNT
UsMjU1LDAuOSkiIHRleHQtYW5jaG9yPSJtaWRkbGUiPlRyYW5zZm9ybSBvcjwvdGV4dD4KICA8dGV
4dCB4PSI0NzAiIHk9IjEyNSIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNp
emU9IjEwIiBmaWxsPSJyZ2JhKDI1NSwyNTUsMjU1LDAuOSkiIHRleHQtYW5jaG9yPSJtaWRkbGUiP
mFnZ3JlZ2F0ZTwvdGV4dD4KCiAgPCEtLSBBcnJvdyAzIC0tPgogIDxsaW5lIHgxPSI1NDAiIHkxPS
IxMDAiIHgyPSI1NzUiIHkyPSIxMDAiIHN0cm9rZT0iIzk0YTNiOCIgc3Ryb2tlLXdpZHRoPSIzIiB
tYXJrZXItZW5kPSJ1cmwoI3BpcGVBcnJvdykiLz4KICA8dGV4dCB4PSI1NTciIHk9IjkwIiBmb250
LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTEiIGZpbGw9IiM2NDc0OGIiI
HRleHQtYW5jaG9yPSJtaWRkbGUiPnw8L3RleHQ+CgogIDwhLS0gT3V0cHV0IHN0YWdlIC0tPgogID
xyZWN0IHg9IjU4MCIgeT0iNjAiIHdpZHRoPSIxNDAiIGhlaWdodD0iODAiIHJ4PSI4IiBmaWxsPSJ
1cmwoI291dHB1dEdyYWQpIiBmaWx0ZXI9InVybCgjcGlwZWxpbmVTaGFkb3cpIi8+CiAgPHRleHQg
eD0iNjUwIiB5PSI5NSIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9I
jEzIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0id2hpdGUiIHRleHQtYW5jaG9yPSJtaWRkbGUiPn
NvcnQgLyBsaW1pdDwvdGV4dD4KICA8dGV4dCB4PSI2NTAiIHk9IjExNSIgZm9udC1mYW1pbHk9IkF
yaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSJyZ2JhKDI1NSwyNTUsMjU1LDAu
OSkiIHRleHQtYW5jaG9yPSJtaWRkbGUiPk9yZGVyIHJlc3VsdHM8L3RleHQ+CgogIDwhLS0gRXhhb
XBsZSBRdWVyeSAtLT4KICA8cmVjdCB4PSI0MCIgeT0iMTYwIiB3aWR0aD0iNzIwIiBoZWlnaHQ9Ij
kwIiByeD0iOCIgZmlsbD0iIzFlMjkzYiIvPgogIDx0ZXh0IHg9IjYwIiB5PSIxODUiIGZvbnQtZmF
taWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTIiIGZpbGw9IiMyMmM1NWUiPmZldGNoPC90ZXh0
PgogIDx0ZXh0IHg9IjEwMCIgeT0iMTg1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpe
mU9IjEyIiBmaWxsPSIjZjhmYWZjIj5zcGFuczwvdGV4dD4KICA8dGV4dCB4PSI2MCIgeT0iMjA1Ii
Bmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEyIiBmaWxsPSIjZjk3MzE2Ij5maWx
0ZXI8L3RleHQ+CiAgPHRleHQgeD0iMTIwIiB5PSIyMDUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2Ui
IGZvbnQtc2l6ZT0iMTIiIGZpbGw9IiNmOGZhZmMiPnNwYW4ua2luZCA9PSA8L3RleHQ+CiAgPHRle
HQgeD0iMjIwIiB5PSIyMDUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTIiIG
ZpbGw9IiNmYmJmMjQiPiJzZXJ2ZXIiPC90ZXh0PgoKICA8dGV4dCB4PSIyOTAiIHk9IjIwNSIgZm9
udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMiIgZmlsbD0iIzk0YTNiOCI+fDwvdGV4
dD4KICA8dGV4dCB4PSIzMDUiIHk9IjIwNSIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1za
XplPSIxMiIgZmlsbD0iI2Y5NzMxNiI+ZmlsdGVyPC90ZXh0PgogIDx0ZXh0IHg9IjM2MCIgeT0iMj
A1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEyIiBmaWxsPSIjZjhmYWZjIj5
kdXJhdGlvbiA+IDwvdGV4dD4KICA8dGV4dCB4PSI0MzAiIHk9IjIwNSIgZm9udC1mYW1pbHk9Im1v
bm9zcGFjZSIgZm9udC1zaXplPSIxMiIgZmlsbD0iI2Y5NzMxNiI+ZmlsdGVyPC90ZXh0PgogIDx0Z
Xh0IHg9IjQzMCIgeT0iMjA1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEyIi
BmaWxsPSIjZjhmYWZjIj5zdW1tYXJpemU8L3RleHQ+CiAgPHRleHQgeD0iNTAwIiB5PSIyMDUiIGZ
vbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTIiIGZpbGw9IiMxNDk2ZmYiPmJ5PC90
ZXh0PgogIDx0ZXh0IHg9IjUyMCIgeT0iMjA1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250L
XNpemU9IjEyIiBmaWxsPSIjZjhmYWZjIj5zZXJ2aWNlLm5hbWU8L3RleHQ+CgogIDx0ZXh0IHg9Ij
YwIiB5PSIyMjUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTIiIGZpbGw9IiM
5NGEzYjgiPnw8L3RleHQ+CiAgPHRleHQgeD0iNzUiIHk9IjIyNSIgZm9udC1mYW1pbHk9Im1vbm9z
cGFjZSIgZm9udC1zaXplPSIxMiIgZmlsbD0iIzRhM2I4Ij58PC90ZXh0PgogIDx0ZXh0IHg9IjM2M
CIgeT0iMjI1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEyIiBmaWxsPSIjOT
RhM2I4Ij58PC90ZXh0PgogIDx0ZXh0IHg9IjM3NSIgeT0iMjI1IiBmb250LWZhbWlseT0ibW9ub3N
wYWNlIiBmb250LXNpemU9IjEyIiBmaWxsPSIjMjJjNTVlIj5zb3J0PC90ZXh0PgogIDx0ZXh0IHg9
IjQxMCIgeT0iMjI1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEyIiBmaWxsP
SIjZjhmYWZjIj5ieSBjb3VudCA8L3RleHQ+CiAgPHRleHQgeD0iNDYwIiB5PSIyMjUiIGZvbnQtZm
FtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTIiIGZpbGw9IiNmOGZhZmMiPnwgPC90ZXh0PgogIDx0ZXh0IHg9IjQ2MCIgeT0iMjI1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEyIiBmaWxsPSIjZjhmYWZjIj5zb3J0IC
8L3RleHQ+CiAgPHRleHQgeD0iNDYwIiB5PSIyMjUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZv

````
bnQtc2l6ZT0iMTIiIGZpbGw9IiMyMmM1NWUiPmRlc2M8L3RleHQ+CgogIDx0ZXh0IHg9IjUwMCIge
T0iMjI1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEyIiBmaWxsPSIjOTRhM2
I4Ij58PC90ZXh0PgogIDx0ZXh0IHg9IjUxNSIgeT0iMjI1IiBmb250LWZhbWlseT0ibW9ub3NwYWN
lIiBmb250LXNpemU9IjEyIiBmaWxsPSIjMjJjNTVlIj5saW1pdDwvdGV4dD4KICA8dGV4dCB4PS
NTUiIHk9IjIyNSIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMiIgZmlsbD0iI
zYwYTVmYSI+MTA8L3RleHQ+Cjwvc3ZnPgo=)



### Key Differences

| Concept | SQL | DQL |
|---------|-----|-----|
| **Arrays** | `('a', 'b')` parentheses | `{"a", "b"}` curly braces |
| **Comparison** | `=` single equals | `==` double equals |
| **String quotes** | `'single quotes'` | `"double quotes"` |
| **NULL checks** | `IS NULL` / `IS NOT NULL` | `isNull()` / `isNotNull()` |
| **Membership** | `IN (...)` | `in(field, {...})` |
| **Grouping** | `GROUP BY` | `by: {...}` in summarize |

---

## 2. Filtering by Service

Filter spans to focus on specific services. Use `dt.entity.service` for
reliable filtering (entity ID), or `service.name` for display name.

> 💡 **Tip:** `dt.entity.service` is always populated and indexed.
`service.name` may not always be available.

```dql
// Filter spans for a specific service using exact match
fetch spans
| filter service.name == "checkout"
| fields start_time, span.name, span.kind, duration
| sort start_time desc
| limit 50
```

```dql
// Use in() with curly braces {} for multiple values (NOT parentheses!)
fetch spans
| filter in(service.name, {"checkout", "payment", "cart"})
| fields start_time, service.name, span.name, span.kind, duration
| sort start_time desc
| limit 100
```
````

```dql
// Count spans by service entity (more reliable)
fetch spans
| filter isNotNull(dt.entity.service)
| summarize {span_count = count()}, by: {dt.entity.service}
| sort span_count desc
| limit 20
```

---

3. Filtering by Span Kind

Filter spans based on their role in the distributed transaction.

⚠️ **IMPORTANT:** `span.kind` values are **lowercase**!

Kind	Description	Use Case
`"server"`	Handles incoming request	Find inbound API calls
`"client"`	Makes outgoing request	Find calls to dependencies
`"internal"`	Internal processing	Find business logic
`"producer"`	Sends async message	Find message publishers
`"consumer"`	Receives async message	Find message consumers

```dql
// Find all SERVER spans (inbound requests)
// Note: "server" is lowercase, not "SERVER"
fetch spans
| filter span.kind == "server"
| fields start_time, service.name, span.name, duration
| sort duration desc
| limit 50
```

```dql
// Find CLIENT spans (outbound calls to dependencies)
fetch spans
| filter span.kind == "client"
| fields start_time, service.name, span.name, duration
| sort duration desc
| limit 50
```

```dql
// Count spans by kind to understand your traffic patterns
fetch spans
| summarize {span_count = count()}, by: {span.kind}
```

````
| sort span_count desc
```

---

## 4. Filtering by Operation Name

Find spans for specific operations or endpoints using the `span.name`
attribute:

```dql
// Find spans for a specific operation/endpoint
fetch spans
| filter contains(span.name, "checkout")
| fields start_time, service.name, trace.id, span.name, duration,
span.status_code
| sort start_time desc
| limit 50
```

```dql
// Find all POST operations (write operations)
fetch spans
| filter startsWith(span.name, "POST")
| fields start_time, service.name, span.name, duration, span.status_code
| sort start_time desc
| limit 50
```

---

## 5. String Matching Functions

DQL provides several string matching functions:

| Function | Description | Example |
|----------|-------------|----------|
| `contains(field, "text")` | Substring match | `contains(span.name, "user")`
|
| `startsWith(field, "text")` | Prefix match | `startsWith(span.name, "GET")`
|
| `endsWith(field, "text")` | Suffix match | `endsWith(url.path, ".json")` |
| `matchesPhrase(field, "pattern")` | Wildcard pattern |
`matchesPhrase(span.name, "GET /api/*")` |
| `in(field, {"a", "b"})` | Multiple values | `in(span.kind, {"server",
"client"})` |

```dql
````

```
// Contains - partial match anywhere in the string
fetch spans
| filter contains(span.name, "Get")
| fields span.name, service.name
| dedup span.name
| limit 20
```

```dql
// startsWith and endsWith - prefix/suffix matching
fetch spans
| filter startsWith(span.name, "GET") or endsWith(span.name, "query")
| fields span.name
| dedup span.name
| limit 20
```

```dql
// matchesPhrase - wildcard pattern matching with *
fetch spans
| filter matchesPhrase(span.name, "GET /api/*")
| fields span.name
| dedup span.name
| limit 20
```

```dql
// Use dedup to see unique span names per service
fetch spans
| filter span.kind == "server"
| fields service.name, span.name
| dedup service.name, span.name
| sort service.name asc
| limit 50
```

---

6. Finding Specific Traces

Locate all spans belonging to a specific trace:

```dql
// First, find some trace IDs to work with
fetch spans
| filter span.kind == "server"
| fields start_time, trace.id, span.name, service.name
| sort start_time desc
```

````
| limit 10
```

```dql
// Find all spans for a specific trace ID
// Replace YOUR_TRACE_ID with an actual trace.id from above
fetch spans
// | filter trace.id == "YOUR_TRACE_ID"
| fields start_time, span.id, span.parent_id, span.name, service.name,
duration
| sort start_time asc
| limit 100
```

```dql
// Find root spans (entry points) - spans without a parent
fetch spans
| filter isNull(span.parent_id)
| fields start_time, trace.id, span.name, service.name, duration,
span.status_code
| sort start_time desc
| limit 50
```

---

## 7. HTTP Span Queries

Query HTTP-specific span attributes for API troubleshooting:

| Attribute | Description |
|-----------|-------------|
| `http.request.method` | HTTP method (GET, POST, etc.) |
| `http.response.status_code` | HTTP status code (200, 404, 500) |
| `http.route` | URL route pattern (use this, not url.path for aggregation) |
| `url.path` | Full URL path (may contain PII) |

```dql
// Query HTTP spans with response status codes
fetch spans
| filter isNotNull(http.response.status_code)
| fields start_time,
         service.name,
         http.request.method,
         http.route,
         http.response.status_code,
         duration
| sort start_time desc
````

````
| limit 100
```

```dql
// Find HTTP 5xx errors (server errors)
fetch spans
| filter http.response.status_code >= 500
      and http.response.status_code < 600
| fields start_time,
         service.name,
         http.request.method,
         http.route,
         http.response.status_code,
         span.status_message,
         duration
| sort start_time desc
| limit 50
```

```dql
// Summarize HTTP status codes by route
fetch spans
| filter isNotNull(http.response.status_code)
| summarize {status_count = count()}, by: {http.response.status_code}
| sort http.response.status_code asc
```

---

## 8. Database Span Queries

Analyze database operations captured as spans:

| Attribute | Description |
|-----------|-------------|
| `db.system` | Database type (mysql, postgresql, redis) |
| `db.name` | Database name |
| `db.operation` | Operation type (SELECT, INSERT, UPDATE) |
| `db.statement` | The database query (may contain sensitive data) |

```dql
// Find all database spans
fetch spans
| filter isNotNull(db.system)
| fields start_time,
         service.name,
         db.system,
         db.name,
````

```
         db.operation,
         duration
| sort duration desc
| limit 50
```

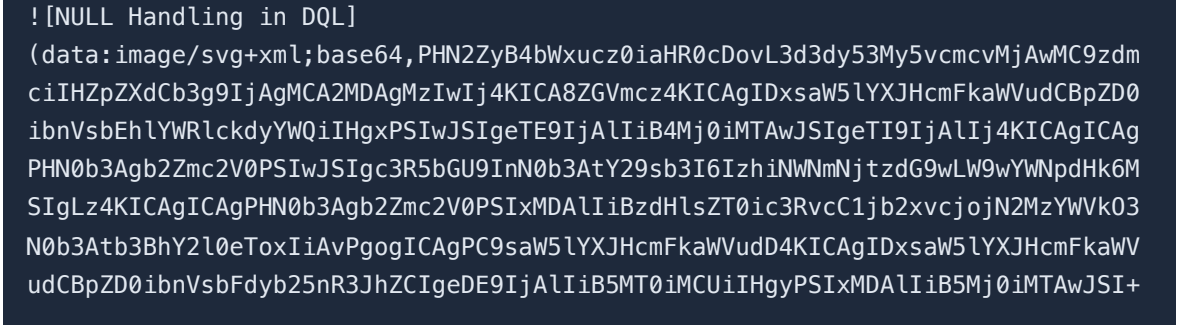
```dql
// Find slow database queries (over 100ms)
fetch spans
| filter isNotNull(db.system)
      and duration > 100ms
| fieldsAdd duration_ms = duration / 1000000
| fields start_time,
         service.name,
         db.system,
         db.operation,
         db.statement,
         duration_ms
| sort duration_ms desc
| limit 50
```

```dql
// Summarize database usage
fetch spans
| filter isNotNull(db.system)
| summarize {
    db_span_count = count(),
    avg_duration_ms = avg(duration) / 1000000
  }, by: {db.system, db.name}
| sort db_span_count desc
```

---

9. Working with NULL Values

⚠️ **DQL uses tri-state boolean logic.** Comparisons with NULL don't work like SQL!



```
```

CiAgICAgIDxzdG9wIG9mZnNldD0iMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiNlZjQ0NDQ7c3RvcC1vc
GFjaXR5OjEiIC8+CiAgICAgIDxzdG9wIG9mZnNldD0iMTAwJSIgc3R5bGU9InN0b3AtY29sb3I6I2
RjMjYyNjtzdG9wLW9wYWNpdHk6MSIgLz4KICAgIDwvbGluZWFyR3JhZGllbnQ+CiAgICA8bGluZWF
yR3JhZGllbnQgaWQ9Im51bGxSaWdodEdyYWQiIHgxPSIwJSIgeTE9IjAlIiB4Mj0iMTAwJSIgeTI9
IjEwMCUiPgogICAgICA8c3RvcCBvZmZzZXQ9IjAlIiBzdHlsZT0ic3RvcC1jb2xvcjojMTBiOTgxO
3N0b3Atb3BhY2l0eToxIiAvPgogICAgICA8c3RvcCBvZmZzZXQ9IjEwMCUiIHN0eWxlPSJzdG9wLW
NvbG9yOiMwNTk2Njk7c3RvcC1vcGFjaXR5OjEiIC8+CiAgICA8L2xpbmVhckdyYWRpZW50PgogICA
gPGZpbHRlciBpZD0ibnVsbFNoYWRvdyI+CiAgICAgIDxmZURyb3BTaGFkb3cgZHg9IjIiIGR5PSIy
IiBzdGREZXZpYXRpb249IjMiIGZsb29kLW9wYWNpdHk9IjAuMTUiLz4KICAgIDwvZmlsdGVyPgogI
DwvZGVmcz4KCiAgPCEtLSBCYWNrZ3JvdW5kIC0tPgogIDxyZWN0IHdpZHRoPSI2MDAiIGhlaWdodD
0iMzIwIiBmaWxsPSIjMGYxNzJhIiByeD0iMTAiLz4KCiAgPCEtLSBIZWFkZXIgLS0+CiAgPHJlY3Q
geD0iMzAiIHk9IjIwIiB3aWR0aD0iNTQwIiBoZWlnaHQ9IjQ1IiByeD0iOCIgZmlsbD0idXJsKCNu
dWxsSGVhZGVyR3JhZCkiIGZpbHRlcj0idXJsKCNudWxsU2hhZG93KSIvPgogIDx0ZXh0IHg9IjUwI
iB5PSI0OCIgZm9udC1mYW1pbHk9InN5c3RlbS11aSwgLWFwcGxlLXN5c3RlbSwgc2Fucy1zZXJpZi
IgZm9udC1zaXplPSIxNiIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IiNmZmYiPk5VTEwgSGFuZGx
pbmcgaW4gRFFMPC90ZXh0PgogIDx0ZXh0IHg9IjU0MCIgeT0iNDgiIGZvbnQtZmFtaWx5PSJzeXN0
ZW0tdWksIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTIiIGZpbGw9InJnYmEoMjU1LDI1NSwyNTUsM
C43KSIgdGV4dC1hbmNob3I9ImVuZCI+Q3JpdGljYWwgS25vd2xlZGdlPC90ZXh0PgoKICA8IS0tIF
dyb25nIHNlY3Rpb24gLS0+CiAgPHJlY3QgeD0iMzAiIHk9IjgwIiB3aWR0aD0iMjYwIiBoZWlnaHQ
9IjEyMCIgcng9IjgiIGZpbGw9IiMxZTI5M2IiIHN0cm9rZT0iI2VmNDQ0NCIgc3Ryb2tlLXdpZHRo
PSIyIi8+CiAgPHJlY3QgeD0iMzAiIHk9IjgwIiB3aWR0aD0iMjYwIiBoZWlnaHQ9IjMwIiByeD0iO
CIgZmlsbD0idXJsKCNudWxsV3JvbmdHcmFkKSIvPgogIDxyZWN0IHg9IjMwIiB5PSIxMDAiIHdpZH
RoPSIyNjAiIGhlaWdodD0iMTAiIGZpbGw9InVybCgjbnVsbFdyb25nR3JhZCkiLz4KICA8dGV4dCB
4PSIxNjAiIHk9IjEwMCIgZm9udC1mYW1pbHk9InN5c3RlbS11aSwgc2Fucy1zZXJpZiIgZm9udC1z
aXplPSIxMiIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IiNmZmYiIHRleHQtYW5jaG9yPSJtaWRkb
GUiPkNvbW1vbiBNaXN0YWtlczwvdGV4dD4KICAgPHRleHQgeD0iNTAiIHk9IjEzNSIgZm9udC1mYW
1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMSIgZmlsbD0iI2ZjYTVhNSI+ZmllbGQgPT0gbnV
sbDwvdGV4dD4KICA8dGV4dCB4PSIyMDAiIHk9IjEzNSIgZm9udC1mYW1pbHk9InN5c3RlbS11aSwg
c2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0iIzk0YTNiOCI+UmV0dXJucyBOVUxMPC90Z
Xh0PgoKICA8dGV4dCB4PSI1MCIgeT0iMTYwIiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LX
NpemU9IjExIiBmaWxsPSIjZmNhNWE1Ij5maWVsZCAhPSBudWxsPC90ZXh0PgogIDx0ZXh0IHg9IjI
wMCIgeT0iMTYwIiBmb250LWZhbWlseT0ic3lzdGVtLXVpLCBzYW5zLXNlcmlmIiBmb250LXNpemU9
IjExIiBmaWxsPSIjOTRhM2I4Ij5SZXR1cm5zIE5VTEw8L3RleHQ+CgogIDx0ZXh0IHg9IjUwIiB5P
SIxODUiIGZvbnQtZmFtaWx5PSJzeXN0ZW0tdWksIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIG
ZpbGw9IiNmODcxNzEiPlRoZXNlIGRvIE5PVCByZXR1cm4gdHJ1ZS9mYWxzZSE8L3RleHQ+CgogIDw
hLS0gUmlnaHQgc2VjdGlvbiAtLT4KICA8cmVjdCB4PSIzMTAiIHk9IjgwIiB3aWR0aD0iMjYwIiBo
ZWlnaHQ9IjEyMCIgcng9IjgiIGZpbGw9IiMxZTI5M2IiIHN0cm9rZT0iIzEwYjk4MSIgc3Ryb2tlL
XdpZHRoPSIyIi8+CiAgPHJlY3QgeD0iMzEwIiB5PSI4MCIgd2lkdGg9IjI2MCIgaGVpZ2h0PSIzMC
Igcng9IjgiIGZpbGw9InVybCgjbnVsbFJpZ2h0R3JhZCkiLz4KICA8cmVjdCB4PSIzMTAiIHk9IjE
wMCIgd2lkdGg9IjI2MCIgaGVpZ2h0PSIxMCIgZmlsbD0idXJsKCNudWxsUmlnaHRHcmFkKSIvPgog
IDx0ZXh0IHg9IjQ0MCIgeT0iMTAwIiBmb250LWZhbWlseT0ic3lzdGVtLXVpLCBzYW5zLXNlcmlmI
iBmb250LXNpemU9IjEyIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0iI2ZmZiIgdGV4dC1hbmNob3
I9Im1pZGRsZSI+Q29ycmVjdCBBcHByb2FjaDwvdGV4dD4KCiAgPHRleHQgeD0iMzMwIiB5PSIxMzU
iIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTEiIGZpbGw9IiM2ZWU3YjciPmlz
TnVsbChmaWVsZCk8L3RleHQ+CiAgPHRleHQgeD0iNDcwIiB5PSIxMzUiIGZvbnQtZmFtaWx5PSJze
XN0ZW0tdWksIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTEiIGZpbGw9IiM5NGEzYjgiPlJldHVybn
MgdHJ1ZSBpZiBudWxsPC90ZXh0PgoKICA8dGV4dCB4PSIzMzAiIHk9IjE2MCIgZm9udC1mYW1pbHk
9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMSIgZmlsbD0iIzZlZTdiNyI+aXNOb3ROdWxsKGZpZWxk

KTwvdGV4dD4KICA8dGV4dCB4PSI0NzAiIHk9IjE2MCIgZm9udC1mYW1pbHk9InN5c3RlbS11aSwgc
2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0iIzk0YTNiOCI+UmV0dXJucyB0cnVlIGlmIE
5PVCBudWxsPC90ZXh0PgoKICA8dGV4dCB4PSIzMzAiIHk9IjE4NSIgZm9udC1mYW1pbHk9InN5c3R
lbS11aSwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzM0ZDM5OSI+VXNlIHRoZXNl
IGZvciBib29sZWFuIGxvZ2ljITwvdGV4dD4KCiAgPCEtLSBFeGFtcGxlIHNlY3Rpb24gLS0+CiAgP
HJlY3QgeD0iMzAiIHk9IjIxNSIgd2lkdGg9IjU0MCIgaGVpZ2h0PSI5MCIgcng9IjgiIGZpbGw9Ii
MxZTI5M2IiIGZpbHRlcj0idXJsKCNudWxsU2hhZG93KSIvPgogIDx0ZXh0IHg9IjUwIiB5PSIyNDA
iIGZvbnQtZmFtaWx5PSJzeXN0ZW0tdWksIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTIiIGZvbnQt
d2VpZ2h0PSJib2xkIiBmaWxsPSIjZjFmNWY5Ij5FeGFtcGxlIFVzYWdlOjwvdGV4dD4KCiAgPHRle
HQgeD0iNTAiIHk9IjI2NSIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMSIgZm
lsbD0iIzk0YTNiOCI+Ly8gRmluZCBzcGFucyB3aXRoIGVycm9yczwvdGV4dD4KICA8dGV4dCB4PSI
1MCIgeT0iMjg1IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjExIiBmaWxsPSIj
YTViNGZjIj5mZXRjaCBzcGFucyB8IGZpbHRlciBpc05vdE51bGwoc3Bhbi5zdGF0dXNfbWVzc2FnZ
Sk8L3RleHQ+Cjwvc3ZnPgo=)

```dql
// Find spans that have database information
fetch spans
| filter isNotNull(db.system)
| summarize {db_span_count = count()}, by: {db.system, db.name}
| sort db_span_count desc
```

```dql
// Find HTTP spans with missing route (potential instrumentation issue)
fetch spans
| filter isNotNull(http.request.method) and isNull(http.route)
| fields service.name, span.name, http.request.method, url.path
| dedup service.name, span.name
| limit 20
```

---

10. Combining Multiple Filters

Build complex queries by combining multiple filter conditions:

> 💡 **Performance Tip:** Apply more restrictive filters first for better query performance.

```dql
// Complex filter: Find slow SERVER spans in the checkout service
fetch spans
| filter service.name == "checkout"
      and span.kind == "server"
```

````
      and duration > 500ms
| fieldsAdd duration_ms = duration / 1000000
| fields start_time,
         span.name,
         duration_ms,
         span.status_code
| sort duration_ms desc
| limit 50
```

```dql
// Find error spans for specific services and operations
fetch spans
| filter span.status_code == "error"
      and in(service.name, {"payment", "checkout"})
      and span.kind == "server"
| fieldsAdd duration_ms = duration / 1000000
| fields start_time,
         service.name,
         span.name,
         span.status_message,
         trace.id,
         duration_ms
| sort start_time desc
| limit 50
```

```dql
// Find spans: either server errors OR slow successful requests
fetch spans
| filter http.response.status_code >= 500
      or (http.response.status_code >= 200
          and http.response.status_code < 300
          and duration > 1s)
| fieldsAdd duration_ms = duration / 1000000
| fields start_time,
         service.name,
         http.request.method,
         http.route,
         http.response.status_code,
         duration_ms
| sort start_time desc
| limit 100
```

---

## Summary
````
````

```
In this notebook, you learned:

✅ **DQL ≠ SQL** - Critical syntax differences (arrays use `{}`, use `==`,
`isNull()`)
✅ **Filter by service** using exact match, `in()`, and pattern matching
✅ **Filter by span kind** - values are lowercase (`"server"`, `"client"`)
✅ **String matching** - `contains()`, `startsWith()`, `endsWith()`,
`matchesPhrase()`
✅ **Find traces** by trace.id and identify root spans
✅ **Query HTTP spans** including status codes, methods, routes
✅ **Analyze database spans** to find slow queries
✅ **Handle NULL values** with `isNull()` / `isNotNull()`
✅ **Use `dedup`** to see unique values
✅ **Combine filters** for complex, precise queries

---

## Next Steps

Continue to **SPANS-03: Trace Analysis & Troubleshooting** to learn:
- Identifying error patterns and failure points
- Latency analysis across services
- Root cause analysis techniques
- Tracing request flows through your system
```