

Table of Contents

| | |
|---------------------------------------|---|
| 1. Query Language Reference | 1 |
| 1.1. Basic Syntax | 1 |
| 1.2. Common Patterns | 2 |
| 1.3. Advanced Examples | 3 |
| 1.4. Variables in Filters | 4 |
| 1.5. Performance Tips | 5 |
| 1.6. Integration with REST APIs | 5 |
| 1.7. Error Handling | 6 |
| 1.8. See Also | 7 |

1. Query Language Reference

Quantum uses an ANTLR-based query language (BIAPIQuery.g4) for filtering, searching, and constraining data across all REST endpoints. This single, consistent syntax works everywhere: list APIs, permission rules, and access resolvers.

1.1. Basic Syntax

Operators

| Operator | Symbol | Example |
|-----------------------|--------|----------------------------------|
| Equals | : | name:"Widget" |
| Not equals | :! | status:! "DELETED" |
| Less than | :< | price:<##100 |
| Greater than | :> | quantity:>#50 |
| Less than or equal | :≤ | createdAt:≤2024-12-31 |
| Greater than or equal | :>= | updatedAt:>=2024-01-01T00:00:00Z |
| Field exists | :~ | description:~ |
| In list | :^ | status:^["ACTIVE" , "PENDING"] |

Value Types

| Type | Prefix | Example |
|------------------|---------------|-------------------------------------|
| String | none or "..." | name:widget or name:"Super Widget" |
| Number (integer) | # | quantity:#42 |
| Number (decimal) | ## | price:##19.99 |
| Date | none | shipDate:2024-12-25 |
| DateTime | none | createdAt:2024-12-25T10:30:00Z |
| Boolean | none | active:true |
| Null | none | description:null |
| ObjectId | none | id:507f1f77bcf86cd799439011 |
| Reference | @@ | parentId:@@507f1f77bcf86cd799439011 |
| Variable | \${...} | ownerId:\${principalId} |

Logical Operators

| Operator | Symbol | Example |
|----------|-------------------------|--|
| AND | <code>&&</code> | <code>active:true && price:>##10</code> |
| OR | <code> </code> | <code>status:"ACTIVE" status:"PENDING"</code> |
| NOT | <code>!!</code> | <code>!(price:<##5)</code> |
| Grouping | <code>()</code> | <code>(active:true featured:true) && price:>##0</code> |

1.2. Common Patterns

String Matching

```
# Exact match
name:"Super Widget"

# Wildcard matching
name:*widget*      # contains "widget"
name:widget*       # starts with "widget"
name:*widget       # ends with "widget"
name:w?dget        # single character wildcard

# Case sensitivity (depends on database collation)
name:"WIDGET"       # may or may not match "widget"
```

Numeric Ranges

```
# Price between 10 and 100
price:>=##10 && price:<=##100

# Quantity greater than 0
quantity:>#0

# Exact count
itemCount:#5
```

Date and Time Queries

```
# Orders from today
createdAt:>=2024-12-25

# Orders from last week
createdAt:>=2024-12-18 && createdAt:<2024-12-25

# Specific timestamp
updatedAt:2024-12-25T14:30:00Z
```

```
# Orders modified this year
updatedAt:>=2024-01-01T00:00:00Z
```

List Membership

```
# Status in specific values
status:^[ "ACTIVE", "PENDING", "PROCESSING" ]

# User IDs from a list
ownerId:^[ "user1", "user2", "user3" ]

# ObjectId list
categoryId:^[ @507f1f77bcf86cd799439011, @507f1f77bcf86cd799439012 ]

# Mixed types (coerced automatically)
priority:^[ #1, #2, #3 ]
```

Null and Existence Checks

```
# Field has any value (not null)
description:~

# Field is null
description:null

# Field is not null
description:!null

# Field exists and is not empty string
description:~ && description:!"
```

1.3. Advanced Examples

Complex Business Logic

```
# Active products under $50 OR featured products at any price
(active:true && price:<##50) || featured:true

# Orders needing attention: overdue OR high-value pending
(dueDate:<2024-12-25 && status:! "COMPLETED") ||
(status:"PENDING" && totalAmount:>##1000)

# Products with inventory issues
(quantity:<=#5 && reorderPoint:>#5) || stockStatus:"OUT_OF_STOCK"
```

Multi-tenant Filtering

```
# User's own records
dataDomain.ownerId:${principalId}

# Organization-wide access
dataDomain.orgRefName:${orgRefName}

# Tenant-scoped with public sharing
dataDomain.tenantId:${pTenantId} || dataDomain.orgRefName:"PUBLIC"
```

Audit and Compliance

```
# Records modified by specific user
auditInfo.lastUpdatedBy:"john.doe"

# Changes in date range
auditInfo.lastUpdatedDate:>=2024-12-01 &&
auditInfo.lastUpdatedDate:<2024-12-31

# Created vs modified
auditInfo.createdDate:auditInfo.lastUpdatedDate # never modified
auditInfo.createdDate:!auditInfo.lastUpdatedDate # has been modified
```

1.4. Variables in Filters

Variables are resolved from the current security context and can be used in permission rules and access resolvers.

Standard Variables

| Variable | Description |
|-----------------------------------|---------------------------------------|
| <code>\${principalId}</code> | Current user's ID |
| <code>\${pTenantId}</code> | Principal's tenant ID |
| <code>\${pAccountId}</code> | Principal's account ID |
| <code>\${pOrgRefName}</code> | Principal's organization |
| <code>\${realm}</code> | Current realm/database |
| <code>\${area}</code> | Current functional area |
| <code>\${functionalDomain}</code> | Current functional domain |
| <code>\${action}</code> | Current action (CREATE, UPDATE, etc.) |

Custom Variables from Access Resolvers

```
// In your AccessListResolver
```

```

@Override
public String key() {
    return "accessibleCustomerIds"; // becomes ${accessibleCustomerIds}
}

@Override
public Collection<?> resolve(...) {
    return Arrays.asList("CUST001", "CUST002", "CUST003");
}

```

```

# Use in filter
customerId:[${accessibleCustomerIds}]

```

1.5. Performance Tips

Efficient Queries

```

# Good: Use indexed fields first
status:"ACTIVE" && createdAt:>=2024-01-01

# Better: Combine with specific values
status:"ACTIVE" && ownerId:${principalId} && createdAt:>=2024-01-01

# Avoid: Leading wildcards on large collections
name:*widget # can be slow on millions of records

```

Projection for Large Objects

```

# In REST calls, limit returned fields
GET /products/list?filter=active:true&projection=+id,+name,+price,-description

```

1.6. Integration with REST APIs

List Endpoints

```

# Basic filtering
GET /products/list?filter=active:true

# With sorting and pagination
GET /products/list?filter=price:>##10&sort=-createdAt&skip=20&limit=10

# Complex filter with projection
GET
/orders/list?filter=(status:"PENDING"||status:"PROCESSING")&&totalAmount:>##100&projec

```

```
tion=+id,+status,+totalAmount,+customerName
```

Permission Rules

```
- name: user-own-records
  priority: 300
  match:
    method: [GET]
    url: /api/**
  effect: ALLOW
  andFilterString: "dataDomain.ownerId:${principalId}"
```

Access Resolvers

```
// Resolver returns customer IDs user can access
public Collection<?> resolve(...) {
    return customerService.getAccessibleIds(principalId);
}

// Used in permission rule
andFilterString: "customerId:^([${accessibleCustomerIds}])"
```

1.7. Error Handling

Common syntax errors and solutions:

```
# Wrong: Missing quotes for multi-word strings
name:Super Widget
# Right:
name:"Super Widget"

# Wrong: Incorrect number prefix
price:19.99
# Right:
price:##19.99

# Wrong: Invalid date format
createdDate:12/25/2024
# Right:
createdDate:2024-12-25

# Wrong: Unbalanced parentheses
(active:true && price:>##10
# Right:
(active:true && price:>##10)
```

1.8. See Also

- [REST CRUD Querying](#)
- [Permission Rules](#)
- [Access Resolvers](#)