RFC-025: Soup Query Language

Status: Draft **Date:** January 2026 **Author:** Derrell Piper ddp@eludom.net
Implementation: Proposed

Abstract

This RFC specifies the query language for the Library of Cyberspace soup: how principals search, filter, and retrieve objects from the content-addressed store using the rich metadata layer. Queries are themselves content-addressed and auditable.

Motivation

The soup contains infinite metadata. Finding needles requires a query language that is:

- **Expressive** - Complex predicates, joins across objects
- **Efficient** - Indexes, bloom filters, query planning
- **Secure** - Queries respect capability boundaries
- **Auditable** - Query patterns reveal access patterns

Newton's soup had cursor-based queries. We extend this with content-addressed semantics.

Query Model

Basic Queries

```
;; Find by type
(soup-query type: 'document)

;; Find by attribute
(soup-query author: "alice@example.com")

;; Find by multiple attributes (AND)
(soup-query type: 'document
            author: "alice@example.com"
            created-after: "2026-01-01")

;; Find by content hash
(soup-query hash: "sha256:7f83b1657ff1fc...")
```

Query Results

```
;; Results are lazy cursors
(define results (soup-query type: 'document))

;; Iterate
(soup-cursor-next results)    ; => soup-object or #f
(soup-cursor-peek results)    ; => next without advancing
(soup-cursor-reset results)   ; => back to beginning

;; Collect (careful with large result sets)
(soup-cursor->list results)   ; => list of soup-objects
(soup-cursor-count results)   ; => total count
```

Predicates

```
;; Comparison operators
(soup-query size: (> 1000000))           ; larger than 1MB
(soup-query created: (< "2025-01-01"))   ; before 2025
(soup-query name: (like "rfc-*"))        ; glob pattern
(soup-query tags: (contains "urgent"))   ; list membership

;; Logical operators
(soup-query (and (type: 'document)
                 (or (author: "alice")
                     (author: "bob"))))

;; Negation
(soup-query (not (type: 'tombstone)))

;; Existence
(soup-query (exists 'encryption-key))    ; has attribute
(soup-query (missing 'expires))          ; lacks attribute
```

Advanced Queries

Range Queries

```
;; Numeric ranges
(soup-query size: (between 1000 10000))

;; Date ranges
(soup-query created: (between "2026-01-01" "2026-12-31"))

;; Lexicographic ranges
```

```
(soup-query name: (between "rfc-010" "rfc-020"))
```

Full-Text Search

```
;; Search indexed content
(soup-query (text-search "capability delegation"))

;; With highlighting
(soup-query (text-search "SPKI certificate")
            highlight: #t)

;; Phrase search
(soup-query (text-search "\"monotonic attenuation\""))

;; Fuzzy search
(soup-query (text-search "delgation~"))  ; typo-tolerant
```

Reference Traversal

```
;; Objects referencing this hash
(soup-query references: "sha256:target...")

;; Objects referenced by this hash
(soup-query referenced-by: "sha256:source...")

;; Transitive closure (careful - can be huge)
(soup-query (transitive-references "sha256:root...")
            max-depth: 3)
```

Temporal Queries

```
;; Objects as they existed at a point in time
(soup-query type: 'document
            as-of: "2026-01-01T00:00:00Z")

;; Objects modified in time window
(soup-query modified: (between "2026-01-01" "2026-01-07"))

;; Version history
(soup-query (versions-of "sha256:current..."))
```

Query Composition

Subqueries

```
;; Objects authored by members of a group
(soup-query author: (soup-query type: 'principal
                                member-of: "engineering"))

;; Documents referencing any RFC
(soup-query type: 'document
            references: (soup-query name: (like "rfc-*")))
```

Aggregation

```
;; Count by type
(soup-aggregate
  group-by: 'type
  aggregate: (count))

;; Total size by author
(soup-aggregate
  group-by: 'author
  aggregate: (sum 'size))

;; Average document size
(soup-aggregate
  where: (type: 'document)
  aggregate: (avg 'size))

;; Distinct values
(soup-aggregate
  aggregate: (distinct 'content-type))
```

Ordering and Pagination

```
;; Sort by creation date, newest first
(soup-query type: 'document
            order-by: '((created desc)))

;; Multi-column sort
(soup-query type: 'document
            order-by: '((author asc) (created desc)))

;; Pagination
(soup-query type: 'document
            order-by: '((created desc))
            limit: 20
```

```
            offset: 40)

;; Cursor-based pagination (more efficient)
(soup-query type: 'document
            order-by: '((created desc))
            after: "sha256:last-seen...")
```

Capability-Aware Queries

Query Filtering

Queries automatically filter results based on the querying principal's capabilities:

```
(define (soup-query-with-caps query principal)
  "Execute query filtered by principal's capabilities"
  (let* ((raw-results (execute-query query))
         (caps (principal-capabilities principal)))
    (filter (lambda (obj)
              (can-read? principal obj caps))
            raw-results)))
```

Capability Queries

```
;; What can I access?
(soup-query (accessible-by (current-principal)))

;; Who can access this?
(soup-query type: 'certificate
            grants-access-to: "sha256:target...")

;; Find my capabilities
(soup-query type: 'certificate
            subject: (current-principal))
```

Query Authorization

```
;; Some queries require explicit capability
(soup-query type: 'audit-log)  ; requires audit-read capability

;; Query capability certificate
(spki-cert
  (issuer vault-admin)
  (subject auditor-key)
  (capability
    (action query)
```

```
    (object (type 'audit-log)))
  (validity (not-after "2027-01-01")))
```

Indexes

Index Types

```
;; B-tree index (ordered, range queries)
(define-index 'created-idx
  type: 'btree
  on: 'created)

;; Hash index (equality only, faster)
(define-index 'hash-idx
  type: 'hash
  on: 'content-hash)

;; Full-text index (search)
(define-index 'content-idx
  type: 'fulltext
  on: 'content
  language: 'english)

;; Composite index
(define-index 'author-date-idx
  type: 'btree
  on: '(author created))

;; Bloom filter index (membership testing)
(define-index 'tags-bloom
  type: 'bloom
  on: 'tags
  false-positive-rate: 0.01)
```

Index Selection

```
(define (plan-query query indexes)
  "Select best index for query"
  (let ((predicates (query-predicates query)))
    (find (lambda (idx)
            (index-covers? idx predicates))
          (sort indexes index-selectivity>))))
```

Index Maintenance

```
;; Indexes are updated on soup-put
(define (soup-put-indexed obj)
  (let ((hash (soup-put obj)))
    (for-each (lambda (idx)
                (index-insert! idx obj hash))
              (applicable-indexes obj))
    hash))

;; Periodic index optimization
(define (optimize-indexes)
  (for-each index-compact! (all-indexes)))
```

Query Execution

Query Planning

```
(define (execute-query query)
  "Plan and execute query"
  (let* ((plan (plan-query query))
         (cost (estimate-cost plan)))
    (when (> cost *query-cost-limit*)
      (error "Query too expensive" cost))
    (execute-plan plan)))

(define (plan-query query)
  "Generate query execution plan"
  `(query-plan
    (predicates ,(query-predicates query))
    (index ,(select-index query))
    (filter ,(remaining-predicates query))
    (order ,(query-order query))
    (limit ,(query-limit query))))
```

Execution Strategies

```
;; Index scan
(define (index-scan index predicate)
  (index-range-query index
                     (predicate-lower-bound predicate)
                     (predicate-upper-bound predicate)))

;; Full scan (last resort)
(define (full-scan predicate)
```

```
  (filter predicate (all-soup-objects)))

;; Index intersection
(define (index-intersect idx1 idx2 pred1 pred2)
  (set-intersection
    (index-scan idx1 pred1)
    (index-scan idx2 pred2)))
```

Query Caching

```
;; Query results can be cached
(define query-cache (make-lru-cache 1000))

(define (cached-query query)
  (let ((key (query->hash query)))
    (or (lru-get query-cache key)
        (let ((result (execute-query query)))
          (lru-put! query-cache key result)
          result))))

;; Cache invalidation on soup changes
(define (invalidate-query-cache obj)
  (let ((affected (queries-affected-by obj)))
    (for-each (lambda (q)
                (lru-remove! query-cache (query->hash q)))
              affected)))
```

Distributed Queries

Federated Query

```
;; Query across multiple vaults
(soup-query type: 'document
            federation: '(vault-a vault-b vault-c))

;; Query with locality preference
(soup-query type: 'document
            prefer-local: #t)
```

Query Routing

```
(define (route-query query vaults)
  "Route query to appropriate vaults"
  (let ((partitions (query-partitions query)))
    (map (lambda (vault)
```

```
          (list vault (subquery-for-vault query vault)))
        (filter (lambda (v)
                  (vault-has-partition? v partitions))
                vaults))))
```

Result Merging

```
(define (merge-results results order)
  "Merge sorted results from multiple vaults"
  (let ((streams (map result->stream results)))
    (merge-sorted-streams streams (order->comparator order))))
```

Query Audit

Audit Trail

```
;; All queries are logged
(define (audited-query query principal)
  (let ((start (current-time))
        (result (soup-query-with-caps query principal)))
    (audit-append
      action: 'query
      principal: principal
      query: (query->sexp query)
      result-count: (soup-cursor-count result)
      duration: (- (current-time) start))
    result))
```

Query Analysis

```
;; Analyze query patterns
(soup-query type: 'audit-entry
            action: 'query
            principal: "alice")

;; Find expensive queries
(soup-query type: 'audit-entry
            action: 'query
            duration: (> 1000))  ; > 1 second

;; Access pattern analysis
(soup-aggregate
  where: (and (type: 'audit-entry) (action: 'query))
  group-by: 'principal
  aggregate: (count))
```

Query Language Grammar

```
query       ::= (soup-query clause*)
clause      ::= attribute-clause | predicate-clause | option-clause
attribute   ::= symbol ':' value
predicate   ::= '(' op value* ')'
op          ::= '>' | '<' | '>=' | '<=' | '=' | '!='
              | 'like' | 'between' | 'contains'
              | 'and' | 'or' | 'not'
              | 'exists' | 'missing'
              | 'text-search' | 'references' | 'referenced-by'
option      ::= 'order-by' | 'limit' | 'offset' | 'after'
              | 'as-of' | 'highlight' | 'federation'
value       ::= string | number | boolean | hash | query
```

Security Considerations

Query Injection

```
;; Never interpolate user input into queries
;; BAD:
(soup-query name: (string-append "rfc-" user-input))

;; GOOD:
(soup-query name: (like (sanitize-pattern user-input)))

(define (sanitize-pattern s)
  (string-map (lambda (c)
                (if (member c '(#\* #\? #\[ #\]))
                    #\\
                    c))
              s))
```

Query Cost Limits

```
;; Prevent denial of service via expensive queries
(define *query-cost-limit* 10000)
(define *query-timeout* 30)  ; seconds

(define (safe-query query)
  (with-timeout *query-timeout*
    (let ((cost (estimate-cost query)))
      (when (> cost *query-cost-limit*)
```

```
        (error "Query exceeds cost limit"))
      (execute-query query))))
```

Information Leakage

```
;; Query existence can leak information
;; Even "not found" reveals something

;; Use constant-time responses for sensitive queries
(define (private-query query)
  (let ((result (soup-query query)))
    (if (authorized? (current-principal) result)
        result
        (constant-time-not-found))))
```

Implementation Notes

Performance

- Index selection is critical for large soups
- Bloom filters for fast negative lookups
- Query result streaming to avoid memory exhaustion
- Connection pooling for federated queries

Compatibility

- Query language inspired by Newton's soup cursors
- SQL-like semantics where applicable
- S-expression syntax for Scheme integration

References

1. Newton Programmer's Guide - Soup and cursor APIs
2. SQLite Query Planner - Query optimization
3. RFC-020: Content-Addressed Storage
4. RFC-021: Capability Delegation

Changelog

- **2026-01-07** - Initial draft

Implementation Status: Draft **Dependencies:** soup, cas, spki **Integration:** Vault queries, federation, audit trail