Query Gateway Ontology Support — Design

Version 1.3.0-SNAPSHOT, 2026-02-17T20:12:18Z

Table of Contents

Implementation status: Section 1 (pass realm/tenant into planning) is implemented: QueryPlanner and MorphiaUtils accept an optional variableMap; QueryGatewayResource builds variableMap from SecurityContext or from the request realm and passes it to convertToPlannedQuery in find() and deleteMany(). Integration test: `QueryGatewayResourceIT.find_with_hasEdge_receives_tenant_context_and_returns_200`.

Delete prevention: Query Gateway delete and deleteMany now enforce the same referential integrity as MorphiaRepo: `DeleteValidationService` runs before each delete (back-reference check for `@TrackReferences` and all `PreDeleteHook` beans, e.g. ontology BLOCK_IF_REFERENCED). On violation the gateway returns 409 (ReferentialIntegrity or DeleteBlocked). deleteMany loads matching entities (capped by `quantum.queryGateway.deleteMany.maxMatches`, default 2000), validates each, then deletes only those ids.

This document describes how to add ontology support to the Query Gateway API (`POST /api/query/find`, `POST /api/query/plan`, and related endpoints) so that callers can use ontology edge predicates (`hasEdge`, `hasOutgoingEdge`, `hasIncomingEdge`, and optionally `hasEdgeAny`, `notHasEdge`) in the query string and get correct, tenant-scoped results.

Chapter 1. Current State

1.1. Query Gateway

- **QueryGatewayResource** (quantum-framework) exposes:
  - `POST /api/query/find` — executes a BIAPI query and returns a Collection envelope (rows, offset, limit).
  - `POST /api/query/plan` — returns mode (FILTER vs AGGREGATION) and expand paths.
  - `GET /api/query/rootTypes`, `POST /api/query/save`, `POST /api/query/delete`, `POST /api/query/deleteMany`.
- For FILTER mode, the gateway calls `MorphiaUtils.convertToPlannedQuery(req.query, root, limit, skip, sortFields)` and then applies `planned.getFilter()` to `ds.find(root)`. No realm or variable map is passed into planning.

1.2. Planner and Filter Building

- **QueryPlanner.plan(query, modelClass, limit, skip, sortFields)** (quantum-morphia-repos):
  - In FILTER mode it calls `MorphiaUtils.convertToFilter(query, modelClass)` — the two-argument overload that passes `null, null` for context (see MorphiaUtils.convertToFilterWContext).
  - In AGGREGATION mode it builds a root filter by calling `MorphiaUtils.convertToFilter(filterOnly, modelClass)` again without context.
- **MorphiaUtils.convertToFilter(queryString, modelClass)** delegates to `convertToFilterWContext(queryString, null, null, modelClass)`. When `pcontext` and `rcontext` are null, the listener is created as `new QueryToFilterListener(modelClass)` — i.e. no PrincipalContext, so no variableMap.
- **QueryToFilterListener** (quantum-morphia-repos) already implements:
  - `enterHasEdgeExpr`, `enterHasOutgoingEdgeExpr`, `enterHasIncomingEdgeExpr` — they read `tenantId` from `variableMap` (keys `pTenantId` or `tenantId`), then use CDI to look up `OntologyEdgeRepo` and call `srcIdsByDst(tenantId, predicate, dst)` (or the inverse for hasIncomingEdge). If `variableMap` is null or `tenantId` is blank, the ontology lookup is skipped and the listener pushes `Filters.eq("_id", "none")` (fail closed, no results).
- Conclusion: When the gateway runs a find with a query that contains `hasEdge(…)`, the filter is built with **no variableMap**, so **tenantId is null** and ontology lookups never run with a valid tenant. Result: hasEdge in the gateway effectively returns no rows for ontology-predicate queries.

1.3. Grammar

- BIAPIQuery.g4 (quantum-models) already defines:
  - `hasEdgeExpr`: `hasEdge(predicate, dst)`

- `hasOutgoingEdgeExpr`: `hasOutgoingEdge(predicate, dst)`
- `hasIncomingEdgeExpr`: `hasIncomingEdge(predicate, src)`

- It does **not** define `hasEdgeAny(predicate, [id1, id2, …])` or `notHasEdge(predicate, dst)`. Those exist only in **ListQueryRewriter** (quantum-ontology-policy-bridge) and are used by OntologyAwareResource, not by the BIAPI parser.

1.4. Documentation

- ai-agent-integration.adoc and QueryHintsProvider state that "for the generic query gateway find, use attribute filters; for ontology-constrained lists, use the resource's ontology list endpoint." So the docs currently direct agents **away** from using ontology predicates in the gateway.

Chapter 2. Goals

1. **Enable ontology edge predicates in the gateway find** — When the request has a valid realm (or principal context with tenantId), `hasEdge`, `hasOutgoingEdge`, and `hasIncomingEdge` in the query string should resolve via OntologyEdgeRepo and return the correct entity set for that tenant.
2. **Single integration point** — Agents and UIs can use `POST /api/query/find` with ontology predicates instead of being directed to per-resource ontology list endpoints when they want relationship-based filtering.
3. **Optional: hasEdgeAny and notHasEdge** — Parity with ListQueryRewriter and OntologyAwareResource constraint semantics (e.g. "invoices for any of these customers", "exclude this org").

Chapter 3. Design

3.1. 1. Pass realm/tenant (or full variable map) into planning (required)

Problem: The gateway never passes realm or PrincipalContext into the planner, so the filter is built with no variableMap and hasEdge gets no tenantId.

Approach:

- **QueryGatewayResource.find(FindRequest req):**
  - Resolve realm as today: `String realm = resolveRealm(req.realm);`
  - Build a **variable map** for planning:
    - If `SecurityContext.getPrincipalContext()` is present, use `MorphiaUtils.createStandardVariableMapFrom(pcontext, resourceContext)` so that pTenantId, tenantId, principalId, etc. are set (and ontology + permission rules behave as in other resources). Pass a minimal ResourceContext for the gateway (e.g. area=integration, functionalDomain=query, action=find).
    - Else, build a minimal map with at least `pTenantId` and `tenantId` set to `realm`, so ontology lookups have a tenant even when the request is not fully authenticated (e.g. server-to-server with realm in the body).
- **MorphiaUtils** (quantum-morphia-repos):
  - Add overloads that accept an optional variable map and pass it into the listener: * **convertToFilter(String queryString, Class<?> modelClass, Map<String, String> variableMap) → if variableMap != null, use convertToFilter(queryString, variableMap, new StringSubstitutor(variableMap), modelClass); else current behavior (null context).** * Existing `convertToFilter(String, Map<String,String>, StringSubstitutor, Class)` already builds a listener with that variableMap; reuse it.
- **QueryPlanner** (quantum-morphia-repos):
  - Add an overload `plan(String query, Class<T> modelClass, Integer limit, Integer skip, List<SortSpec.Field> sortFields, Map<String, String> variableMap)`.
  - In FILTER mode: call `MorphiaUtils.convertToFilter(query, modelClass, variableMap)` instead of `convertToFilter(query, modelClass)`.
  - In AGGREGATION mode: when building the root filter from the stripped query, call `convertToFilter(filterOnly, modelClass, variableMap)` so that any hasEdge in the filter part is also tenant-scoped.
- **MorphiaUtils**:
  - Add `convertToPlannedQuery(String query, Class<T> modelClass, Integer limit, Integer skip, List<SortSpec.Field> sortFields, Map<String, String> variableMap)` that calls `planner.plan(…, variableMap)`.
- **QueryGatewayResource**:

- In `find()`, build variableMap as above, then call `MorphiaUtils.convertToPlannedQuery(req.query, root, limit, skip, sortFields, variableMap)` (or a new overload that takes variableMap). In `deleteMany()`, do the same when building the planned query for the delete filter.

Result: hasEdge / hasOutgoingEdge / hasIncomingEdge in the gateway find query will receive a tenantId and OntologyEdgeRepo will return the correct source IDs for that tenant. No change to QueryToFilterListener ontology logic; only the gateway and planner must pass the map through.

3.2. 2. Optional: Add hasEdgeAny and notHasEdge to BIAPI grammar and listener

Goal: Align gateway query syntax with ListQueryRewriter and OntologyAwareResource (e.g. "receivables for any of these customers", "orders not in this org").

- **BIAPIQuery.g4** (quantum-models):
  - Add rules, e.g.: * **hasEdgeAnyExpr: HASEDGEANY LPAREN predicate=(STRING|QUOTED_STRING|VARIABLE) COMMA dstList=valueListExpr RPAREN; notHasEdgeExpr: NOTHASEDGE LPAREN predicate=(STRING|QUOTED_STRING|VARIABLE) COMMA dst=(STRING|QUOTED_STRING|VARIABLE|OID|REFERENCE) RPAREN;** *
  - Add tokens: `HASEDGEANY: 'hasEdgeAny'; NOTHASEDGE: 'notHasEdge';`
  - Add to allowedExpr: `hasEdgeAnyExpr | notHasEdgeExpr`.
- **QueryToFilterListener** (quantum-morphia-repos):
  - Implement `enterHasEdgeAnyExpr`: resolve tenantId from variableMap; for each value in dstList, call OntologyEdgeRepo.srcIdsByDst(tenantId, predicate, dst) and take the union of IDs; push `Filters.in("_id", objectIds)` (or `Filters.eq("_id", "none")` if empty). Reuse the same CDI/reflection pattern as hasEdge so that quantum-ontology-mongo remains an optional dependency.
  - Implement `enterNotHasEdgeExpr`: resolve tenantId; get source IDs for that (predicate, dst); then push `Filters.nin("_id", objectIds)` (entities whose id is **not** in the set that have the edge). If the set is empty, do not restrict (all entities pass the notHasEdge clause).
- Regenerate ANTLR lexer/parser after grammar changes; add/update unit tests for the new expressions.

Priority: Can be done in a follow-up once realm/variableMap wiring (design 1) is in place and hasEdge works in the gateway.

3.3. 3. Documentation updates

- **planner-and-query-gateway.adoc**:
  - In "Request/Response: /api/query/find", state that the query string may include ontology edge predicates: `hasEdge(predicate, dst)`, `hasOutgoingEdge(predicate, dst)`, `hasIncomingEdge(predicate, src)`. When the application includes quantum-ontology-mongo

and the request has a valid realm (or principal context with tenantId), these predicates restrict results to entities that have the corresponding ontology relationship in the current tenant. If realm/tenant is missing or ontology is not wired, hasEdge-style predicates evaluate to no results (fail closed).

- **ai-agent-integration.adoc** and **QueryHintsProvider**:
  - Update the ontologyEdges / didYouKnow text to say that the **query gateway find** supports hasEdge / hasOutgoingEdge / hasIncomingEdge when a realm (or principal) is present; agents can use these in the same query string as attribute filters and expand(...). Optionally mention hasEdgeAny and notHasEdge if implemented.
- **ontology.adoc** (or rest-crud):
  - Add a short subsection that the Query Gateway (`POST /api/query/find`) honors ontology edge predicates in the BIAPI query when the request supplies a realm or the principal context has a tenantId; OntologyEdgeRepo (quantum-ontology-mongo) must be on the classpath.

3.4. 4. Optional: Ontology context via gateway

- Some clients may want "ontology edges for entity X" without calling a resource-specific endpoint. Options:
  - (a) Do not add a gateway endpoint; keep using `GET /{basePath}/id/{id}/ontology` per OntologyAwareResource.
  - (b) Add `GET /api/query/ontologyContext?rootType=⋯&id=⋯` (or POST with body) that resolves rootType to a collection, looks up OntologyContextEnricher (or OntologyEdgeRepo), and returns the same shape as the existing ontology context response. This requires the gateway (or a shared service) to depend on quantum-ontology-mongo and to know how to get DataDomain from the request (realm/principal). Low priority unless we want a single discovery surface for agents.

Recommendation: Omit (b) for the first iteration; document that ontology **context** (edges for one entity) remains on OntologyAwareResource detail endpoints.

Chapter 4. Dependencies and Backward Compatibility

- **quantum-framework** already depends on quantum-morphia-repos. It does not depend on quantum-ontology-mongo or quantum-ontology-policy-bridge. QueryToFilterListener uses CDI + reflection to resolve OntologyEdgeRepo and OntologyAliasResolver, so when those beans are not present, hasEdge fails closed (no results). No new compile dependency is required for the gateway; applications that want ontology in the gateway must include quantum-ontology-mongo (and typically quantum-ontology-policy-bridge) as today.
- Passing a variableMap from the gateway into the planner is backward compatible: when variableMap is null, behavior stays as today (convertToFilter(query, modelClass) with no context). Existing callers of convertToPlannedQuery without variableMap are unchanged.

Chapter 5. Implementation Order

1. **Phase 1 — Realm/variableMap in gateway (required):**
   - MorphiaUtils: add `convertToFilter(query, modelClass, variableMap)` (delegate to existing overload when variableMap != null).
   - QueryPlanner: add `plan(…, Map<String, String> variableMap)` and use convertToFilter with variableMap in both FILTER and AGGREGATION paths.
   - MorphiaUtils: add `convertToPlannedQuery(…, Map<String, String> variableMap)`.
   - QueryGatewayResource.find(): build variableMap (from PrincipalContext or minimal map from resolveRealm), call convertToPlannedQuery(…, variableMap).
   - QueryGatewayResource.deleteMany(): same when building the filter for the delete query.
   - Test: integration test that POST /api/query/find with query containing hasEdge(predicate, dst) and valid realm returns expected rows when ontology edges exist.
2. **Phase 2 — Docs:**
   - Update planner-and-query-gateway.adoc, ai-agent-integration.adoc, QueryHintsProvider, and ontology/rest-crud as in section 3.
3. **Phase 3 — Optional hasEdgeAny / notHasEdge:**
   - Grammar + listener + tests; then document.
4. **Phase 4 — Optional ontology context endpoint:** Only if required; otherwise keep context on OntologyAwareResource.

Chapter 6. Summary

| Item | Action | |------|--------| | **Realm/tenant in planning** | Gateway builds variableMap (from PrincipalContext or minimal { pTenantId, tenantId } = realm) and passes it into MorphiaUtils.convertToPlannedQuery / QueryPlanner.plan. MorphiaUtils and QueryPlanner gain overloads that accept variableMap and pass it to convertToFilter. | | **hasEdge in find** | No change to QueryToFilterListener; once variableMap contains tenantId, existing hasEdge/hasOutgoingEdge/hasIncomingEdge logic works. | | **hasEdgeAny / notHasEdge** | Optional: add to BIAPI grammar and QueryToFilterListener so gateway find supports the same semantics as ListQueryRewriter. | | **Documentation** | State that gateway find supports ontology edge predicates when realm/tenant is set and quantum-ontology-mongo is on the classpath; update agent hints. | | **Ontology context** | Leave on OntologyAwareResource; no gateway endpoint in first iteration. |

This design adds ontology support to the query gateway API in a minimal, backward-compatible way by threading the existing realm/principal context into the filter-building path so that ontology edge predicates already implemented in QueryToFilterListener receive a tenantId and behave correctly.