The **Query Engine** in your Agentic Procurement SaaS platform is the **core system that translates structured prompt inputs into executable queries** over your **knowledge graph**, **SQL-like datastores**, or **vector search layers**.

It must be designed for:

* 🔎 **Interpretability** (users can see what was queried)
* 🧠 **Extensibility** (LLMs, KG, SQL, vector search)
* ✅ **Policy enforcement** (via the Policy Engine)
* 🔁 **Interactivity** (LLM agents may refine, retry, or chain queries)

**🧱 Query Engine – Core Structure**

**📁 High-Level Folder Architecture**

query\_engine/

├── \_\_init\_\_.py

├── query\_engine\_service.py

├── dsl\_parser.py

├── graph\_query\_builder.py

├── sql\_query\_builder.py

├── vector\_query\_builder.py

├── policy\_validator.py

├── query\_executor.py

├── result\_formatter.py

└── models.py

**🧠 Key Responsibilities by Module**

| **Module** | **Role** |
| --- | --- |
| query\_engine\_service.py | Entry point. Routes prompt → query → execution → response |
| dsl\_parser.py | Converts prompt intent + entities into a structured intermediate query model |
| graph\_query\_builder.py | Builds SPARQL or GraphQL queries from DSL |
| sql\_query\_builder.py | Builds SQL queries (e.g., PostgreSQL, Redshift) |
| vector\_query\_builder.py | Constructs vector-based queries for embedding similarity (e.g., contract clause similarity) |
| policy\_validator.py | Enforces row/column-level security & data access policies |
| query\_executor.py | Executes query on the right backend (KG, SQL DB, vector index) |
| result\_formatter.py | Formats results for tabular display, agent response, or next-agent routing |
| models.py | Common schema: QueryIntent, FilterClause, ExecutionPlan, QueryResult |

**📄 Example Data Flow**

**Step-by-Step:**

1. **Prompt Engine Output**:

{

"intent": "supplier\_contract\_filter",

"entities": {

"spend\_gt": 100000,

"esg\_score": "low",

"contract\_end\_before": "2025-10-01"

}

}

1. **DSL Parser Output (Intermediate Query Model)**:

{

"target": "supplier",

"filters": [

{ "field": "contract.spend", "operator": ">", "value": 100000 },

{ "field": "supplier.esg\_score", "operator": "=", "value": "low" },

{ "field": "contract.end\_date", "operator": "<", "value": "2025-10-01" }

]

}

1. **Graph Query Builder → SPARQL**:

SELECT ?supplier ?spend ?endDate ?esgScore

WHERE {

?supplier :hasContract ?contract .

?contract :spend ?spend .

?contract :endDate ?endDate .

?supplier :esgScore ?esgScore .

FILTER(?spend > 100000 && ?esgScore = "low" && ?endDate < "2025-10-01")

}

1. **Policy Validator**:

* User is not allowed to see supplier.internal\_rating → field redacted or removed from query

1. **Query Executor**:

* Dispatches to KG backend or SQL DB
* Optional: cache check, retry, fallback agent trigger

1. **Result Formatter**:

* Converts result into JSON table, UI block, or formatted message for LLMDraftingAgent

**🧠 Execution Modes Supported**

| **Mode** | **Use Case** |
| --- | --- |
| **Graph (SPARQL/GraphQL)** | Rich relationships: contracts, suppliers, ESG, multi-hop |
| **SQL** | Spend, invoice, PO line items, ERP-integrated data |
| **Vector Search** | Clause similarity, past negotiation matching, RFP alignment |
| **Hybrid** | Combine multiple sources and merge via QueryAggregatorAgent |

**🔐 Security Layer**

Integrated with your **Policy Engine**:

* ✅ Row/column-level access control (e.g., finance team sees more than a business user)
* ✅ Sensitive fields blocked or masked
* ✅ Auto-logs every query for audit

**🚀 Optional Additions**

| **Feature** | **Purpose** |
| --- | --- |
| **Natural Language ↔ DSL debugger** | Let users “see” what their input translated into |
| **Query Cache** | Cache expensive KG joins for repeated queries |
| **LLM post-processor** | Agent uses query results to recommend actions or next steps |
| **Explainability metadata** | Show why a supplier was included/excluded in filtered result |