

Eugene Patent Count — End-to-End Flow

Developer walkthrough: HTTP → module-load DI → Neo4j COUNT Cypher → {search_key, patent_count} JSON

Audience

Engineers who need a file-referenced trace of the patent-count endpoints. These routes are the cheap companions to the full `/patents/*` search endpoints — clients call the count first to compute the maximum page number, then page through the search. Every reference uses the form `path/to/file.py:line`.

Reference endpoints

- GET `/count/patents/drugs?drug_id=DB00993`
- GET `/count/patents/clinicaltrials?nct_id=NCT05648006`
- GET `/count/patents/geneproteins?gene_protein=ABL1`

All three are declared in `src/foundation/router/patent_count_router.py` and registered by `src/eugene_ws.py:49 / src/eugene_ws.py:71` inside `add_routers(app)`.

Relationship to the patent-search route

The count routes share the `Neo4jPatentQueryAdapter` singleton with `patent_search_router.py` (both call `neo4j_patent_query_adapter()` at module load — `patent_count_router.py:28` and `patent_search_router.py:34`). They walk the same edges with the same anchors and just substitute `count(n2) AS count` for the paginated projection. If a drug returns 17 patents from the count route, the search route with `page_size=50` will return exactly those 17 rows on page 1.

How to read this guide

- Section 0 — schema (the three edge types the count walks). Read first.
- Sections 1-3 — HTTP, Cypher, response shape, top-down.
- Section 4 — pointer to `EUGENE_PATENT_FLOW_GUIDE.pdf` for ETL.
- Section 5 — copy-paste debug walk.
- Section 6 — flat reference index.

0. Schema (read this first)

The patent-count routes anchor on one of three node types and traverse a single hop to a `:Patent_Application` node. There is no APOC traversal, no variable-length match — just one labeled edge.

```
(:drug { node_id }) -[:disclosed_in]- (:Patent_Application)
(:ClinicalTrial { nct_id }) -[:supports_patent_application]-(:Patent_Application)
(:gene_protein { node_name }) -[:patent_app_target]- (:Patent_Application)
```

- Node labels come from `FoundationalNodeEnum` in `src/foundation/model/foundational_node_enum.py`: `DRUG` (line 14, label `drug`), `CLINICAL_TRIAL` (line 10, label `ClinicalTrial`), `GENE_PROTEIN` (line 18, label `gene_protein`), and `PATENT_APPLICATION` (line 31, label `Patent_Application`). Note the mixed case — `Patent_Application` and `ClinicalTrial` keep their PascalCase in Neo4j, the rest are lowercase.
- Relationship types come from `FoundationalRelationshipEnum` in `src/foundation/model/foundational_relationship_enum.py`: `DISCLOSED_IN` (line 25), `SUPPORTS_PATENT_APPLICATION` (line 26), `PATENT_APP_TARGET` (line 27). The drug shape `(:drug)-[:disclosed_in]-(:Patent_Application)` is the same one the `patent_search_router` walks for paginated results.
- The `:disclosed_in` edge is **undirected** in the Cypher (`-[r:`disclosed_in`]-` not `-[r:`disclosed_in`]->`). The count is symmetric — direction in the graph does not change the answer.
- Anchor keys differ per route: drug anchors on `n.node_id`, clinical trial on `n.nct_id` (not `node_id`!), and gene/protein on `n.node_name` (also not `node_id`).
- Like the rest of Eugene there are no Neo4j `CREATE CONSTRAINTS`; uniqueness is by ETL convention, not enforced at the DB layer.

1. HTTP entry — the FastAPI router

Wiring

`src/eugene_ws.py:49` imports the module as `patent_count_router`; `src/eugene_ws.py:71` adds it to the includes list inside `add_routers(app)`. The module-level router = `APIRouter(prefix="/count/patents", tags=["patents"])` (`patent_count_router.py:23-26`) means the final paths are `/count/patents/drugs`, `/count/patents/clinicaltrials`, and `/count/patents/geneproteins`.

Module-load dependency injection

`src/foundation/router/patent_count_router.py:28`

```
patent_query_adapter = neo4j_patent_query_adapter()
```

The DI factory at `src/foundation/conf/conf.py:199-211` resolves to a single `Neo4jPatentQueryAdapter` bound to the shared `_neo4j_driver()` singleton. The same instance is reused by `patent_search_router.py:34` — both routers share one adapter, one driver, and one connection pool.

Endpoint signatures

```
@router.get("/drugs", response_model_exclude_none=True) # line 31
async def count_related_patents_by_drug_id(drug_id):
```

```
@router.get("/clinicaltrials", response_model_exclude_none=True) # line 49
async def count_related_patents_by_clinicaltrail_id(nct_id):
```

```
@router.get("/geneproteins", response_model_exclude_none=True) # line 67
async def count_related_patents_by_geneprotein_id(gene_protein):
```

Query params & validators

- `drug_id` (line 33-42) — `Query(example="DB00993", max_length=64, min_length=1) + AfterValidator(validate_value.validate_value(validate_util.py:35-42))` rejects any of `% _ $ ; : ^ *`. Note: it does *not* enforce the DB prefix — that stricter check lives in `validate_drug_id` (lines 58-68) which this route does not use.
- `nct_id` (line 51-60) — `max_length=24, min_length=1 + AfterValidator(validate_clinical_trail_id(validate_util.py:45-55))`, which delegates to `validate_value` *and* additionally requires the NCT prefix (case-insensitive).
- `gene_protein` (line 69-78) — `max_length=24, min_length=1 + AfterValidator(validate_gene_protein(validate_util.py:71-77))`; same special-char check, no prefix requirement.
- All three routes **require** their query parameter (no defaults); Pydantic returns a 422 if missing.

Normalization

Each handler calls the matching `case_helper` normalizer before hitting the adapter: `normalize_drug_id(drug_id)` (line 44), `normalize_clinical_trial_id(nct_id)` (line 62), and `normalize_gene_protein(gene_protein)` (line 80). All three live in `src/foundation/model/case_helper.py`; they exist so casing on the wire (e.g. `db00993` vs. `DB00993`) does not miss the Neo4j anchor.

First breakpoint: `patent_count_router.py:45, 63, or 81` (the adapter call).

2. Query adapter — the COUNT Cypher

Neo4jPatentQueryAdapter lives at `src/foundation/infra/db/adapter/neo4j_patent_query_adapter.py`. Each count entry point opens a session, begins a transaction, calls the matching private helper, and returns the integer.

Public entry points

- `count_related_patents_by_drug(drug_id) -> int` (lines 43-52).
- `count_related_patents_by_clinicaltrail(nct_id) -> int` (lines 69-80).
- `count_related_patents_by_gene(gene) -> int` (lines 97-108).
- Each opens a session, runs a single read transaction (`session.begin_transaction()`), and the inner helper does `tx.run(search, params).single()`.

Cypher — /count/patents/drugs (lines 248-257)

```
MATCH (n:`drug`)-[r:`disclosed_in`]- (n2:`Patent_Application`)
WHERE n.node_id = $drug_id
RETURN count(n2) as count
```

Cypher — /count/patents/clinicaltrials (lines 282-291)

```
MATCH (n:`ClinicalTrial`)-[r:`supports_patent_application`]- (n2:`Patent_Application`)
WHERE n.nct_id = $nct_id
RETURN count(n2) as count
```

Cypher — /count/patents/geneproteins (lines 316-325)

```
MATCH (n:`gene_protein`)-[r:`patent_app_target`]- (n2:`Patent_Application`)
WHERE n.node_name = $gene
RETURN count(n2) as count
```

COUNT semantics worth noting

- `count(n2)` counts *matched rows*, not distinct `n2` nodes. In this schema every (anchor)-[r]-(Patent_Application) edge is one row, so the number equals the number of edges from the anchor — which in practice is the number of distinct patents because the ETL writes one edge per (anchor, patent) pair. If you ever load duplicate edges, the count will be inflated — use `count(DISTINCT n2)` if you suspect that has happened.
- The relationship is undirected (`-[r]-` not `-[r]->`); the count is symmetric with respect to edge direction in Neo4j.
- If the anchor node does not exist, `count(n2)` returns 0 (not null) — the route returns `{"drug": ..., "patent_count": 0}` rather than 404. The result is `None` guard in the helper (e.g. line 241-242) is defensive; it would only fire if the driver returned no rows at all, which `count(...)` does not.
- Each helper catches (`DriverError`, `Neo4jError`), logs, and re-raises — Neo4j outages surface as a FastAPI 500.
- Labels and relationship strings are interpolated into the Cypher via `%s` substitution from `FoundationalNodeEnum.value[1]` / `FoundationalRelationshipEnum.value[1]` — the *only* Cypher parameter is the anchor id (`$drug_id` / `$nct_id` / `$gene`).

3. Response model — just a dict

Unlike the patent *search* route (which has a typed `PatentSearchResult` Pydantic model and a `PatentSearchResultMapper`) the count route has neither. Each handler returns a plain Python dict; FastAPI serializes it directly. The decorator uses `response_model_exclude_none=True` but **no** `response_model=` class, so the body shape is whatever the handler returns.

Verbatim return statements

```
# patent_count_router.py:46
return {"drug": drug_id, "patent_count": count}

# patent_count_router.py:64
return {"nct_id": nct_id, "patent_count": count}

# patent_count_router.py:82
return {"gene_protein": gene_protein, "patent_count": count}
```

Example JSON

```
GET /count/patents/drugs?drug_id=DB00993
-> { "drug": "DB00993", "patent_count": 17 }

GET /count/patents/clinicaltrials?nct_id=NCT05648006
-> { "nct_id": "NCT05648006", "patent_count": 3 }

GET /count/patents/geneproteins?gene_protein=ABL1
-> { "gene_protein": "ABL1", "patent_count": 42 }
```

Implications

- The search-key field name differs per route (`drug` vs. `nct_id` vs. `gene_protein`). Clients cannot use a single key name to read the echo — if you are building a generic UI, branch on the route.
- There is no Pydantic schema in `src/foundation/model/patent/` for this response, and the OpenAPI doc will show only `{}` (unknown object) for the body. If this becomes painful, lift the shape into e.g. `PatentCountResponse(BaseModel)` and pass `response_model=PatentCountResponse`.
- Page-math convention: clients divide `patent_count` by the search route's `page_size` (max 100) and ceil to get the max page. The docstring on `patent_search_router.py` `page` param calls this endpoint out by reference for exactly that purpose.

4. ETL pointer — how the patents get into Neo4j

The count routes are read-only; they assume the `:Patent_Application` nodes and their `:disclosed_in / :supports_patent_application / :patent_app_target` edges already exist. The ingest chain is the USPTO + LLM pipeline already documented in a separate guide.

Cross-reference

See docs/EUGENE_PATENT_FLOW_GUIDE.pdf (generated by `generate_patent_flow_guide.py`) for the full end-to-end ETL: USPTO bulk-data download, per-application LLM extraction of drug/disease/gene mentions, deduplication, and the upsert into Neo4j that builds the very subgraph this count route reads. That guide also covers seed data and how organizational ownership (`:OWNS` / assignee edges) is layered on after the patents are loaded.

Quick orientation

- Upstream input: USPTO bulk JSON dumps in `resources/data/patent/`.
- LLM extraction step produces per-patent assertions linking each `:Patent_Application` to anchor nodes (`:drug`, `:ClinicalTrial`, `:gene_protein`) by their already-canonical `node_id/nct_id/node_name`.
- Upsert is idempotent via `MERGE` on patent number; re-running the ingest does not double-count.
- If the count for a known drug looks low, the likely cause is an upstream LLM extraction miss, *not* a bug in this route. Verify with the Cypher in Section 2 run directly in `cypher -shell`.

5. Suggested debugging walk

- Start the service locally: `uvicorn src.eugene_ws:app --reload`. Confirm Neo4j is up (`docker compose ps eugene-neo4j`) and that the patent subgraph is loaded (`MATCH (p:`Patent_Application`) RETURN count(p);`).
- Hit each endpoint with curl: `curl "http://localhost:8080/count/patents/drugs?drug_id=DB00993"`, then the `clinicaltrials` and `geneproteins` variants. Expect a JSON dict with `patent_count` as an integer.
- Breakpoint at `patent_count_router.py:45` (the adapter call). Step into `Neo4jPatentQueryAdapter.count_related_patents_by_drug` (`neo4j_patent_query_adapter.py:43`). Inspect the assembled Cypher string at `neo4j_patent_query_adapter.py:230-247`.
- Copy that Cypher into `cypher-shell` and run with `:param drug_id => "DB00993"`. The integer must match the HTTP response. If it does not, the only candidate explanations are (a) normalization in `normalize_drug_id` mutated the id, or (b) you are running against a different Neo4j instance than the API is.
- Force the validator: `curl ".../count/patents/drugs?drug_id=DB%2400993"` (URL-encoded \$). Expect a 422 from `validate_value`. Repeat for the `clinicaltrials` route with a missing NCT prefix; expect a 422 from `validate_clinical_trail_id`.
- Cross-check with the search route: `curl ".../patents/drugs?drug_id=DB00993&page=1&page_size=100"` and confirm `len(results) == patent_count` (assuming `patent_count <= 100`). Divergence indicates a duplicate edge or a `DISTINCT` mismatch worth investigating.

6. Reference index (file paths)

Route & wiring

```
src/eugene_ws.py:49,71      (import + add_routers)
src/foundation/router/patent_count_router.py
src/foundation/router/validate_util.py:35-42  (validate_value)
src/foundation/router/validate_util.py:45-55  (validate_clinical_trail_id)
src/foundation/router/validate_util.py:71-77  (validate_gene_protein)
src/foundation/model/case_helper.py           (normalize_*)
src/foundation/conf/conf.py:199-211           (neo4j_patent_query_adapter)
```

Adapter (Cypher)

```
src/foundation/infra/db/adapter/neo4j_patent_query_adapter.py
:43-52  count_related_patents_by_drug
:69-80  count_related_patents_by_clinicaltrail
:97-108 count_related_patents_by_gene
:225-257 _count_related_patents_by_drug + Cypher generator
:259-291 _count_related_patents_by_clinicaltrail + Cypher generator
:293-325 _count_related_patents_by_gene + Cypher generator
```

Domain model (labels & rel types)

```
src/foundation/model/foundational_node_enum.py
:10  CLINICAL_TRIAL      = ClinicalTrial
:14  DRUG                = drug
:18  GENE_PROTEIN        = gene_protein
:31  PATENT_APPLICATION  = Patent_Application
src/foundation/model/foundational_relationship_enum.py
:25  DISCLOSED_IN        = disclosed_in
:26  SUPPORTS_PATENT_APPLICATION = supports_patent_application
:27  PATENT_APP_TARGET    = patent_app_target
```

Companion search route (same adapter, shares Cypher anchors)

```
src/foundation/router/patent_search_router.py
src/foundation/mapper/patent_search_result_mapper.py
src/foundation/model/patent/patent_search_result.py
src/foundation/model/patent/patent_and_gene_search_result.py
```

ETL (cross-reference)

```
docs/EUGENE_PATENT_FLOW_GUIDE.pdf
generate_patent_flow_guide.py
```

End of patent-count route flow guide.