Detailed Explanation: Closure Tables and Graph Traversal

Understanding the Mechanics of Ontology-Driven Queries in Snowflake

RDF Knowledge Graph Proof-of-Concept | Technical Deep Dive

Table of Contents

1. Introduction: The Problem of Graph Traversal

Biological knowledge is inherently hierarchical. A hepatocyte is a type of epithelial cell, which is a type of cell. This simple fact creates a fundamental challenge for data analysis.

Consider a scientist who asks: *"Show me drug efficacy data for all epithelial-derived cancers."* In their experimental database, they have data labeled with specific cell types like "hepatocyte," "mammary epithelial cell," or "pancreatic ductal cell." None of these records are explicitly tagged as "epithelial." Yet biologically, they all share epithelial lineage.

To answer this question, the system must traverse a hierarchy—starting from "epithelial cell" and discovering all its descendants in the ontology. This is the core problem that closure tables solve.

The Fundamental Challenge

In a relational database, hierarchies are stored as parent-child relationships (edges). Finding all descendants requires following these edges recursively—a computationally expensive operation when performed at query time. For deep hierarchies with thousands of nodes, this can make queries unacceptably slow.

Why Not Just Use Recursive Queries?

Modern SQL databases support recursive Common Table Expressions (CTEs) that can traverse hierarchies. However, there are significant drawbacks to performing recursion at query time:

- **Performance:** Each query must repeat the recursive computation, even if the hierarchy hasn't changed
- **Complexity:** Every query that needs lineage information must include complex recursive SQL
- **Unpredictability:** Query time depends on hierarchy depth, which varies unpredictably
- **AI Limitations:** AI-powered query generators (like Cortex Analyst) struggle to produce correct recursive SQL from natural language

The solution is to precompute the transitive closure—a table that stores all ancestor-descendant relationships explicitly. This transforms expensive recursive traversal into simple table joins.

2. Foundation: The Node-Edge Data Model

Before understanding closure tables, we must understand how the knowledge graph stores biological knowledge in two fundamental tables.

The KG_NODE Table: Entities

Every concept in the ontology becomes a node. This includes cell types, genes, anatomical structures, and biological processes. Each node has:

Column	Purpose	Example
`NODE_ID`	Unique identifier (URI or curie)	CL:0000182
`NODE_TYPE`	Category of entity	CellType
`NAME`	Human-readable label	hepatocyte
`DESCRIPTION`	Definition from ontology	A liver parenchymal cell...

EXAMPLE NODES FROM CELL ONTOLOGY

```
NODE_ID         NODE_TYPE    NAME
─────────────────────────────────────────────────────
CL:0000000      CellType     cell
CL:0000066      CellType     epithelial cell
CL:0000182      CellType     hepatocyte
CL:0002327      CellType     mammary epithelial cell
CL:0000083      CellType     epithelial cell of pancreas
CL:0000084      CellType     T cell
CL:0000236      CellType     B cell
```

The KG_EDGE Table: Relationships

Relationships between nodes are stored as directed edges. The most important relationship types for our purposes are:

- **subClassOf:** Indicates that one entity is a specialized type of another (hepatocyte subClassOf epithelial cell)
- **BFO_0000050 (part_of):** Indicates compositional relationships (liver lobe part_of liver)
- **regulates:** Indicates functional relationships between biological processes

Column	Purpose	Example
`EDGE_ID`	Unique identifier	E_00001
`SRC_ID`	Source node (the child/part)	CL:0000182
`DST_ID`	Destination node (the parent/whole)	CL:0000066
`EDGE_TYPE`	Type of relationship	subClassOf

EXAMPLE EDGES FORMING A HIERARCHY

```
SRC_ID (Child)      EDGE_TYPE     DST_ID (Parent)
──────────────────────────────────────────────────────
CL:0000182          subClassOf    CL:0000066       (hepatocyte → epithelial cell)
CL:0002327          subClassOf    CL:0000066       (mammary epithelial → epithelial cell)
CL:0000066          subClassOf    CL:0000000       (epithelial cell → cell)
CL:0000084          subClassOf    CL:0000542       (T cell → lymphocyte)
CL:0000542          subClassOf    CL:0000738       (lymphocyte → leukocyte)
```

Key Insight: Direct vs. Transitive Relationships

The KG_EDGE table stores only *direct* relationships—one hop. To know that hepatocyte is ultimately derived from cell requires following two edges: hepatocyte → epithelial cell → cell. This multi-hop traversal is what closure tables precompute.

3. Closure Tables: Theory and Purpose

A closure table is a precomputed structure that stores all reachable ancestor-descendant pairs, eliminating the need for recursive traversal at query time.

What Is Transitive Closure?

In graph theory, the **transitive closure** of a directed graph is a new graph where there exists an edge from node A to node C if there is a path of any length from A to C in the original graph.

If A → B and B → C in the original graph,
then A → B, B → C, ***and*** *A → C exist in the transitive closure.*

Applied to biological hierarchies: if hepatocyte is a subclass of epithelial cell, and epithelial cell is a subclass of cell, then the transitive closure includes the relationship hepatocyte → cell, even though no direct edge exists.

Structure of a Closure Table

Our closure tables store three pieces of information for each ancestor-descendant pair:

Column	Purpose	Example
`CHILD_NODE_ID`	The descendant (more specific term)	CL:0000182 (hepatocyte)
`ANCESTOR_NODE_ID`	The ancestor (more general term)	CL:0000066 (epithelial cell)
`HOPS`	Number of edges in the path	1

CONCEPTUAL VIEW: FROM EDGES TO CLOSURE

```
ORIGINAL EDGES (KG_EDGE)              CLOSURE TABLE (KG_CELLTYPE_ANCESTORS)
════════════════════════              ═════════════════════════════════════

hepatocyte ──────────┐                CHILD          ANCESTOR          HOPS
                     ▼                ─────────────────────────────────────────
              epithelial cell         hepatocyte     hepatocyte         0  ← self
                     │                hepatocyte     epithelial cell    1  ← direct
                     ▼                hepatocyte     cell               2  ← transitive
                   cell               epithelial     epithelial cell    0  ← self
                                      epithelial     cell               1  ← direct
                                      cell           cell               0  ← self
```

Why Include Self-References (HOPS = 0)?

Each node is added as its own ancestor with HOPS = 0. This serves an important purpose: when joining data products to the closure table, exact matches are automatically included. If a cell line is explicitly labeled "epithelial cell" (not a subtype), querying for "epithelial cell and descendants" will include it because the self-reference row exists.

The HOPS Column

The HOPS value enables nuanced queries. You can ask for "direct children only" (HOPS = 1) or "all descendants within 3 levels" (HOPS ≤ 3). This provides control over how broadly the system interprets lineage relationships.

4. How Closure Tables Are Calculated

The closure table is built using a recursive Common Table Expression (CTE) that iteratively discovers ancestors at increasing depths.

The Algorithm in Plain Language

1. **Base Case:** Start with all direct parent-child relationships from the edge table. Each child is 1 hop from its direct parent.
2. **Recursive Step:** For each relationship discovered so far, look at the parent's parents. Add these grandparent relationships with hop count incremented by 1.
3. **Termination:** Continue until no new ancestors are discovered, or a maximum depth limit is reached (we use 10 hops as a safety limit).
4. **Self-References:** Add a row for each node pointing to itself with 0 hops.
5. **Deduplication:** Remove duplicate rows (the same child-ancestor pair may be discovered via different paths).

Detailed Walkthrough: Building Cell Type Closure

Let's trace through exactly how the closure table is built for a small hierarchy:

SAMPLE HIERARCHY

```
                       cell (CL:0000000)
                           │
                 ┌─────────┴──────────┐
                 │                    │
        epithelial cell        leukocyte
        (CL:0000066)          (CL:0000738)
                 │                    │
      ┌──────────┼──────────┐         │
      │          │          │         │
 hepatocyte  mammary    pancreatic  lymphocyte
(CL:0000182)  epithelial  cell     (CL:0000542)
             (CL:0002327) (CL:0000083)    │
                                         │
                                      T cell
                                    (CL:0000084)
```

Iteration 1: Direct Relationships (HOPS = 1)

The algorithm first queries all direct subClassOf edges between CellType nodes:

CHILD	ANCESTOR	HOPS
hepatocyte	epithelial cell	1
mammary epithelial	epithelial cell	1
pancreatic cell	epithelial cell	1
epithelial cell	cell	1
leukocyte	cell	1
lymphocyte	leukocyte	1
T cell	lymphocyte	1

Iteration 2: Grandparents (HOPS = 2)

For each row from iteration 1, the algorithm looks up whether the ANCESTOR has its own parent. It joins the discovered ancestors back to the edge table:

CHILD	ANCESTOR	HOPS	How Discovered
hepatocyte	cell	2	hepatocyte → epithelial → cell
mammary epithelial	cell	2	mammary → epithelial → cell
pancreatic cell	cell	2	pancreatic → epithelial → cell
lymphocyte	cell	2	lymphocyte → leukocyte → cell
T cell	leukocyte	2	T cell → lymphocyte → leukocyte

Iteration 3: Great-Grandparents (HOPS = 3)

CHILD	ANCESTOR	HOPS	How Discovered
T cell	cell	3	T cell → lymphocyte → leukocyte → cell

At this point, no new ancestors are discoverable (we've reached the root "cell" for all paths), so recursion terminates.

Final Step: Add Self-References

Every CellType node gets a row pointing to itself:

CHILD	ANCESTOR	HOPS
cell	cell	0
epithelial cell	epithelial cell	0
hepatocyte	hepatocyte	0
... (all other cell types)		

The Complete Closure for Hepatocyte

After building the closure table, querying "all ancestors of hepatocyte" returns three rows: hepatocyte itself (0 hops), epithelial cell (1 hop), and cell (2 hops). This information is available instantly via a simple SELECT—no recursion needed at query time.

5. Connecting Closure Tables to Data Products

The closure table's value is realized when it bridges biological knowledge to experimental data through foreign key relationships and lineage-aware views.

The Foreign Key Bridge

Experimental data products store a reference to the ontology via a NODE_ID foreign key. For example, the CELL_LINE_CATALOG table includes:

Column	Purpose	Example Value
`CELL_LINE_ID`	Primary key	CL_001
`CELL_LINE_NAME`	Lab name	HepG2
`TISSUE_ORIGIN`	Tissue type	Liver
`CELL_TYPE_NODE_ID`	**Link to ontology**	CL:0000182

The `CELL_TYPE_NODE_ID` column contains the URI of the ontology term that describes this cell line's cell type. For HepG2 (a liver cancer cell line), this points to "hepatocyte" in the Cell Ontology.

The Lineage View Pattern

A lineage view joins experimental data to the closure table, producing a **flattened representation** where each data record appears multiple times—once for each of its ancestors.

HOW THE LINEAGE VIEW JOIN WORKS

```
CELL_LINE_CATALOG                    KG_CELLTYPE_ANCESTORS
════════════════════                 ════════════════════════════════
CELL_LINE_ID: CL_001                 CHILD_NODE_ID   ANCESTOR_NODE_ID   HOPS
CELL_LINE_NAME: HepG2                ─────────────────────────────────────────
CELL_TYPE_NODE_ID: CL:0000182   ───► CL:0000182      CL:0000182          0
                                     CL:0000182      CL:0000066          1
                                     CL:0000182      CL:0000000          2


                  JOIN ON
           CELL_TYPE_NODE_ID = CHILD_NODE_ID
                    │
                    ▼

V_CELL_LINE_CELLTYPE_LINEAGE (Result)
════════════════════════════════════════════════════════════════════════════
CELL_LINE_ID  CELL_LINE_NAME  CELL_TYPE_ANCESTOR     HOPS
────────────────────────────────────────────────────────────────────────────
CL_001        HepG2           hepatocyte              0     ← exact type
CL_001        HepG2           epithelial cell         1     ← parent
CL_001        HepG2           cell                    2     ← grandparent
```

This single cell line now appears in three rows—one for each level of its ancestry. When a query asks for "epithelial-derived cell lines," HepG2 will be found because one of its CELL_TYPE_ANCESTOR values is "epithelial cell."

Composing Views for Complex Queries

The architecture builds multiple levels of views that compose together:

VIEW COMPOSITION HIERARCHY

```
Layer 1: Base Tables
┌────────────────────┐  ┌────────────────────┐  ┌────────────────────┐
│  CELL_LINE_CATALOG │  │  KG_NODE           │  │  KG_CELLTYPE_      │
│  (experimental)    │  │  (ontology terms)  │  │  ANCESTORS         │
└────────────────────┘  └────────────────────┘  └────────────────────┘
         │                        │                        │
         └───────────┬────────────┴────────────────────────┘
                     │
Layer 2: Cell Line with Lineage
         ┌───────────▼───────────────────────────────────┐
         │      V_CELL_LINE_CELLTYPE_LINEAGE             │
         │  Each cell line × all its ancestor types      │
         └───────────────────────────────────────────────┘
                              │
Layer 3: Efficacy with Lineage
         ┌────────────────────▼──────────────────────────┐
         │      V_IN_VIVO_EFFICACY_LINEAGE               │
         │  Efficacy studies with treatment + model +    │
         │  ALL ancestor cell types for the model        │
         └───────────────────────────────────────────────┘
```

The Power of Precomputation

Because the closure table precomputes all transitive relationships, the lineage views are simple JOINs—no recursion required. This means:

- Queries are fast and predictable
- The semantic view can expose these joins to Cortex Analyst
- Query complexity doesn't increase with hierarchy depth

6. The Semantic View: Enabling Cortex Analyst

The native Snowflake Semantic View (PHARMA_KG_SEMANTIC_VIEW) is the bridge that enables Cortex Analyst to understand your data and generate correct SQL from natural language.

What Is a Semantic View?

A Semantic View is a **native Snowflake object** (not a YAML file) that defines the logical structure of your data for AI-powered querying. It contains metadata about:

- **Tables:** Which tables/views the AI can query
- **Dimensions:** Categorical columns for filtering and grouping
- **Facts:** Numeric columns for aggregation and measurement
- **Relationships:** How tables join together (foreign keys)
- **Synonyms:** Alternative terms users might use (drug = treatment = compound)

> **Why Native Semantic View Instead of YAML?**
>
> Native Snowflake Semantic Views enforce relationship validity at creation time. Primary keys and foreign keys are explicitly declared and validated. This prevents misconfigured joins that could produce incorrect query results.

How the Semantic View Exposes Closure Tables

The key innovation is that the semantic view includes the closure table (KG_CELLTYPE_ANCESTORS) as a **joinable table** with defined relationships:

Semantic Name	Physical Table	Purpose
cell_lines	V_CELL_LINE_ONTOLOGY	Cell line catalog with ontology links
cell_type_lineage	KG_CELLTYPE_ANCESTORS	Precomputed cell type ancestry for traversal
ontology_nodes	KG_NODE	Human-readable names for ontology terms

Relationship Definitions

The semantic view defines how these tables connect, enabling Cortex Analyst to generate correct JOINs:

RELATIONSHIP CHAIN FOR LINEAGE QUERIES

```
┌─────────────┐                  ┌──────────────────┐
│  cell_lines │                  │ cell_type_lineage│
│             │  JOIN ON         │                  │
│ CELL_TYPE_  │─────────────────►│ CHILD_NODE_ID    │
│ NODE_ID     │                  │                  │
└─────────────┘                  │ ANCESTOR_NODE_ID │──┐
                                 │                  │  │
                                 │ HOPS             │  │
                                 └──────────────────┘  │
                                                       │
                                      JOIN ON          │
                                 ┌─────────────────────┘
                                 │
                                 ▼
                           ┌──────────────┐
                           │ ontology_nodes│
                           │              │
                           │ NODE_ID      │
                           │ NAME         │  ← Human-readable
                           │ DESCRIPTION  │     ancestor name
                           └──────────────┘
```

With these relationships defined, Cortex Analyst knows: *"To find a cell line's ancestors, join cell_lines to cell_type_lineage on CELL_TYPE_NODE_ID = CHILD_NODE_ID, then join to ontology_nodes to get the human-readable ancestor name."*

Synonyms for Natural Language Understanding

The semantic view includes synonyms that help Cortex Analyst understand different ways users might phrase the same concept:

Column	Synonyms
DRUG_NAME	drug, compound, treatment
TUMOR_GROWTH_INHIBITION	TGI, efficacy, tumor inhibition
RESPONSE_CATEGORY	response, RECIST
MODEL_TYPE	model type, CDX, PDX

How the Semantic View Enables Graph Traversal

The key insight is that the semantic view exposes the **closure table as a joinable table**. Cortex Analyst doesn't need to understand recursion or graph theory—it simply knows that joining cell_lines to cell_type_lineage gives access to ancestor information. The complexity of graph traversal is hidden behind simple relational joins.

7. Cortex Search Service: Semantic Discovery

The Cortex Search Service (ONTOLOGY_SEARCH_SERVICE) enables users to find ontology concepts by meaning rather than exact text match—a critical capability when scientists use imprecise or unfamiliar terminology.

What Is Cortex Search?

Cortex Search is Snowflake's semantic search capability that indexes text for **similarity-based retrieval**. Unlike traditional SQL WHERE clauses that require exact matches, Cortex Search finds concepts based on meaning.

The Ontology Search Service

We created a Cortex Search Service that indexes all 158,395 ontology terms from the KG_NODE table:

Property	Value
Service Name	ONTOLOGY_SEARCH_SERVICE
Indexed Table	KG_NODE
Searchable Columns	NAME, DESCRIPTION
Returned Columns	NODE_ID, NAME, NODE_TYPE, DESCRIPTION
Total Indexed Terms	158,395

Why Semantic Search Matters

Scientists often use imprecise language when asking questions. Consider these scenarios:

SCENARIO 1: SYNONYM USAGE

User asks: "liver cells"

Ontology term: "hepatocyte"

Semantic search finds "hepatocyte" because it understands the relationship.

SCENARIO 2: PARTIAL KNOWLEDGE

User asks: "cells that make antibodies"

Ontology term: "plasma cell" (description: "a B cell that secretes antibodies")

Semantic search matches based on description content.

How Search Results Enable Queries

When a user asks about an unfamiliar concept, the Cortex Agent can use the Search Service to identify the correct ontology term before querying data:

SEARCH-ASSISTED QUERY FLOW

```
User Question: "Show efficacy for immune cells"
                         │
                         ▼
┌──────────────────────────────────────────────────────────────┐
│  CORTEX SEARCH: "immune cells"                               │
│  ───────────────────────────────────────                     │
│  Results:                                                    │
│    1. leukocyte (CL:0000738) - "white blood cell"            │
│    2. lymphocyte (CL:0000542) - "a leukocyte..."             │
│    3. T cell (CL:0000084) - "lymphocyte of immune..."        │
└──────────────────────────────────────────────────────────────┘
                         │
                         │ Agent selects "leukocyte" as
                         │ the relevant ancestor
                         ▼
┌──────────────────────────────────────────────────────────────┐
│  CORTEX ANALYST: Query with identified term                  │
│  ───────────────────────────────────────                     │
│  "Find efficacy where ANCESTOR_NAME = 'leukocyte'"           │
│  → Returns data for T cells, B cells, etc.                   │
└──────────────────────────────────────────────────────────────┘
```

Search as a Discovery Tool

Cortex Search transforms vague user intent into precise ontology references. Once the correct NODE_ID is identified, it can be used in queries against the closure table to find all related data—combining semantic understanding with structured data access.

8. The Cortex Agent: Intelligent Orchestration

The Cortex Agent (PHARMA_KG_AGENT) is the intelligent layer that decides which tools to use—Analyst for structured queries, Search for concept discovery—and combines their results into coherent answers.

Agent Architecture

The Cortex Agent is configured with two primary tools:

Tool	Type	Purpose
PharmaKGAnalyst	cortex_analyst_text_to_sql	Converts natural language to SQL using the semantic view
OntologySearch	cortex_search	Semantic search over 158K ontology terms

How the Agent Decides Which Tool to Use

The agent is configured with orchestration instructions that guide its decision-making:

AGENT DECISION LOGIC

```
Question Type                              Tool Selection
═══════════════════════════════════════════════════════════════════════

"Show drug efficacy for PDX models"    →   PharmaKGAnalyst (structured)
"What is the average TGI for ADCs?"    →   PharmaKGAnalyst (aggregation)
"Find cell lines with BRCA1 knockout"  →   PharmaKGAnalyst (filtering)

"What is a hepatocyte?"                →   OntologySearch (definition)
"Find cell types related to immunity"  →   OntologySearch (discovery)

"Show efficacy for epithelial-derived  →   OntologySearch (identify term)
 cancers"                                  THEN PharmaKGAnalyst (query)

"Compare ADC response in blood vs      →   PharmaKGAnalyst (complex query
 epithelial cancers"                       using closure table directly)
```

Tool Resources Configuration

Each tool is connected to specific Snowflake resources:

Tool	Resource	Configuration
PharmaKGAnalyst	PHARMA_KG_SEMANTIC_VIEW	Warehouse: KG_POC_WH, Timeout: 60s
OntologySearch	ONTOLOGY_SEARCH_SERVICE	Max results: 10, Returns: NODE_ID, NAME, DESCRIPTION

Agent Response Synthesis

The agent doesn't just return raw results—it synthesizes a coherent response that explains biological relationships:

> Example Agent Response
>
> **User:** "Show efficacy for epithelial-derived cancers"
>
> **Agent:** "I found 12 studies across liver, breast, lung, and pancreatic models. These are all epithelial-derived because:
>
> - Hepatocytes (liver) are a type of epithelial cell
> - Mammary epithelial cells (breast) are epithelial
> - Lung adenocarcinoma cells derive from lung epithelium
>
> The average tumor growth inhibition across these models was 67.3%..."

The Agent's Key Value

The Agent bridges the gap between how scientists think (biological concepts, imprecise language) and how data is stored (structured tables, precise identifiers). It translates intent into action and explains results in biological context.

9. Complete Query Flow: End-to-End Example

Let's trace exactly what happens when a scientist asks: "Show drug efficacy for all epithelial-derived cancers."

Step 1: Agent Receives Question

The Cortex Agent receives the natural language question and analyzes the intent:

- **Domain:** Drug efficacy (connects to in_vivo_efficacy table)
- **Filter concept:** "epithelial-derived" (requires lineage traversal)
- **Scope:** "cancers" (models with disease context)

The agent determines this is a **structured data query with lineage requirements**—it will use the Analyst tool, which has access to the closure table.

Step 2: (Optional) Cortex Search for Term Clarification

If the concept "epithelial" is ambiguous or unfamiliar, the Agent may first use Cortex Search:

SEARCH RESULTS FOR "EPITHELIAL"

```
ONTOLOGY_SEARCH_SERVICE Query: "epithelial cell type"

Results:
┌─────────────────────────────────────────────────────────────────┐
│ NODE_ID       NAME                  NODE_TYPE   DESCRIPTION     │
├─────────────────────────────────────────────────────────────────┤
│ CL:0000066    epithelial cell       CellType    A cell that is  │
│                                                 usually found   │
│                                                 lining surfaces │
├─────────────────────────────────────────────────────────────────┤
│ CL:0002076    endo-epithelial cell  CellType    An epithelial   │
│                                                 cell that...    │
└─────────────────────────────────────────────────────────────────┘

Agent selects: CL:0000066 (epithelial cell) as the target ancestor
```

Step 3: Cortex Analyst Generates SQL

Cortex Analyst consults the semantic view and generates SQL. The key is using the closure table to find all descendants of "epithelial cell":

GENERATED SQL (CONCEPTUAL STRUCTURE)

```
-- Step A: Find all cell types descended from 'epithelial cell'
WITH epithelial_descendants AS (
    SELECT DISTINCT child.CHILD_NODE_ID
    FROM KG_CELLTYPE_ANCESTORS child
    JOIN KG_NODE ancestor ON child.ANCESTOR_NODE_ID = ancestor.NODE_ID
    WHERE ancestor.NAME = 'epithelial cell'
)

-- Step B: Find cell lines with those cell types
, matching_cell_lines AS (
    SELECT cl.CELL_LINE_ID
    FROM CELL_LINE_CATALOG cl
    JOIN epithelial_descendants ed
      ON cl.CELL_TYPE_NODE_ID = ed.CHILD_NODE_ID
)

-- Step C: Get efficacy data for those cell lines
SELECT
    iv.MODEL_NAME,
    iv.TISSUE_ORIGIN,
    iv.DISEASE_CONTEXT,
    iv.DRUG_NAME,
    iv.TUMOR_GROWTH_INHIBITION AS TGI,
    iv.RESPONSE_CATEGORY
FROM V_IN_VIVO_EFFICACY iv
WHERE iv.CELL_LINE_ID IN (SELECT CELL_LINE_ID FROM matching_cell_lines)
ORDER BY iv.TUMOR_GROWTH_INHIBITION DESC
```

Step 4: Query Execution and Data Flow

Let's trace the data through each step:

CTE 1: epithelial_descendants

The closure table is queried to find all nodes that have "epithelial cell" as an ancestor:

CHILD_NODE_ID	Why Included
CL:0000066 (epithelial cell)	Self-reference (HOPS = 0)
CL:0000182 (hepatocyte)	Direct child (HOPS = 1)
CL:0002327 (mammary epithelial)	Direct child (HOPS = 1)
CL:0000083 (pancreatic cell)	Direct child (HOPS = 1)

CTE 2: matching_cell_lines

The cell line catalog is filtered to only those whose cell type appears in the epithelial descendants:

CELL_LINE_ID	CELL_LINE_NAME	CELL_TYPE_NODE_ID
CL_001	HepG2	CL:0000182 (hepatocyte)
CL_004	MCF7	CL:0002327 (mammary epithelial)
CL_007	PANC-1	CL:0000083 (pancreatic cell)

Final Result

MODEL_NAME	TISSUE	DISEASE	DRUG	TGI%	RESPONSE
HepG2	Liver	Hepatocellular carcinoma	ADC-X1	78.5	PR
MCF7	Breast	Breast adenocarcinoma	ADC-X1	65.2	SD
PANC-1	Pancreas	Pancreatic adenocarcinoma	Small-M1	45.1	SD

Step 5: Agent Synthesizes Response

The Agent takes the query results and formulates a natural language response that explains the biological context:

The Key Achievement

The scientist asked about "epithelial-derived cancers" and received data from Liver, Breast, and Pancreas models—even though none of these records explicitly mention "epithelial." The system used biological knowledge encoded in the ontology, made accessible through the closure table and semantic view, to discover that hepatocytes, mammary epithelial cells, and pancreatic cells all share epithelial lineage.

10. Summary and Key Insights

The architecture solves the problem of biological hierarchy traversal through a carefully designed pipeline of precomputation, abstraction, and AI orchestration.

The Complete Architecture

FULL SYSTEM ARCHITECTURE

```
┌──────────────────────────────────────────────────────────────────┐
│                                                                  │
│  LAYER 1: GRAPH STORAGE                                          │
│  ══════════════════                                              │
│  KG_NODE stores entities (158K nodes)                            │
│  KG_EDGE stores direct relationships (409K edges)                │
│  Purpose: Faithful representation of ontology structure          │
│                                                                  │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  LAYER 2: PRECOMPUTED TRAVERSAL                                  │
│  ═════════════════════════                                       │
│  KG_CELLTYPE_ANCESTORS contains all ancestor-descendant pairs    │
│  KG_ANATOMY_PARTS_OF contains anatomical part-of relationships   │
│  Purpose: Transform recursive traversal into relational lookup   │
│                                                                  │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  LAYER 3: DATA PRODUCTS                                          │
│  ══════════════════                                              │
│  CELL_LINE_CATALOG, TREATMENTS, IN_VIVO_STUDIES, etc.            │
│  Linked to ontology via NODE_ID foreign keys                     │
│  Purpose: Store experimental results with biological context     │
│                                                                  │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  LAYER 4: SEMANTIC VIEW                                          │
│  ═════════════════════                                           │
│  PHARMA_KG_SEMANTIC_VIEW (native Snowflake object)               │
│  Defines tables, relationships, dimensions, facts, synonyms      │
│  Purpose: Enable Cortex Analyst to generate correct SQL          │
│                                                                  │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  LAYER 5: AI SERVICES                                            │
│  ══════════════════                                              │
│   ┌───────────────────┐   ┌──────────────────────┐              │
│   │ ONTOLOGY_SEARCH_  │   │ CORTEX ANALYST via   │              │
│   │ SERVICE           │   │ Semantic View        │              │
│   │ (semantic discovery)│ │ (structured queries) │              │
│   └─────────┬─────────┘   └──────────┬───────────┘              │
│             │                        │                           │
│             └────────────┬───────────┘                           │
│                          │                                       │
│                          ▼                                       │
│             ┌───────────────────┐                                │
│             │ PHARMA_KG_AGENT   │                                │
│             │ (orchestration)   │                                │
│             └───────────────────┘                                │
│  Purpose: Intelligent routing and response synthesis             │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

Key Insights

INSIGHT 1: PRECOMPUTATION

The transitive closure is computed once when the ontology is loaded. This expensive recursive operation (building ~54,000 paths for cell types) happens offline, not during user queries.

INSIGHT 2: SELF-REFERENCES

Including HOPS = 0 rows ensures that exact matches are always found. Querying for "hepatocyte lineage" includes hepatocyte itself, not just its ancestors.

INSIGHT 3: FOREIGN KEY BRIDGE

Data products link to the ontology via NODE_ID foreign keys. This simple design decision enables powerful cross-domain queries without complex integration.

INSIGHT 4: SEMANTIC VIEW EXPOSURE

The native Semantic View exposes the closure table as a joinable table. Cortex Analyst generates JOINs—it doesn't need to understand graph traversal.

INSIGHT 5: SEMANTIC SEARCH

Cortex Search enables discovery when users don't know exact terminology. Finding "immune cells" returns leukocyte, lymphocyte, T cell—terms that can then be used in structured queries.

INSIGHT 6: AGENT ORCHESTRATION

The Cortex Agent intelligently routes between Search (concept discovery) and Analyst (data queries), combining their capabilities for complex questions.

What Makes This Work in Practice

1. **Performance:** Queries that would require recursive CTEs now use simple indexed JOINs, returning in sub-second time regardless of hierarchy depth.
2. **Simplicity:** Scientists ask natural language questions. The complexity of ontology traversal is invisible to them.
3. **Discovery:** Cortex Search enables exploration when scientists don't know exact terminology, bridging imprecise language to precise ontology terms.
4. **Accuracy:** Biological relationships are captured from authoritative ontologies (Cell Ontology, Gene Ontology), ensuring scientifically correct lineage information.
5. **Maintainability:** When ontologies are updated, the closure table is rebuilt automatically. Data products don't need to change.

Assets Summary

Component	Asset Name	Purpose
Graph Storage	KG_NODE, KG_EDGE	Store ontology entities and relationships
Closure Tables	KG_CELLTYPE_ANCESTORS, KG_ANATOMY_PARTS_OF	Precomputed transitive relationships
Data Products	CELL_LINE_CATALOG, TREATMENTS, IN_VIVO_STUDIES	Experimental data with ontology links
Lineage Views	V_CELL_LINE_CELLTYPE_LINEAGE, V_IN_VIVO_EFFICACY_LINEAGE	Pre-joined data with ancestry
Semantic View	PHARMA_KG_SEMANTIC_VIEW	Query interface for Cortex Analyst
Search Service	ONTOLOGY_SEARCH_SERVICE	Semantic search over 158K terms
AI Agent	PHARMA_KG_AGENT	Intelligent orchestration layer

Final Thought

The architecture transforms a hard problem (recursive graph traversal with semantic understanding) into a solved problem (relational JOINs with AI-powered natural language access). By investing computation at data load time (closure tables) and metadata at design time (semantic view), the system delivers instant, intelligent results at query time—while hiding all complexity behind a conversational interface that scientists can use without any knowledge of graph theory, SQL, or ontology structure.

Document: Detailed Explanation of Closure Tables and Graph Traversal

Project: RDF Knowledge Graph Proof-of-Concept

Assets: PHARMA_KG_SEMANTIC_VIEW | ONTOLOGY_SEARCH_SERVICE | PHARMA_KG_AGENT

Generated: January 2026