Schema Browser component details

**1. Parsing and Standardization:**

Extracting raw metadata from disparate database systems (e.g., PostgreSQL, Hive, Oracle) and converting it into a unified, canonical metadata format. Each DB system exposes its metadata in different formats and structures (e.g., INFORMATION\_SCHEMA for MySQL vs. dba\_tables in Oracle), so this stage transforms and aligns the metadata schema.

A metadata extractor queries system catalogs or APIs to collect data such as table names, columns, data types, primary keys, and constraints. The parser maps these into a standard metadata model (e.g., Data Catalog JSON format or custom schema).

Different databases have different dictionary views (Postgres information\_schema, Hive Metastore, Oracle DBA\_TABLES).

Hive returns column types like STRING or BIGINT, while Oracle may return VARCHAR2 or NUMBER(10,2). The parser converts both to a unified model, like string and decimal, so downstream components can process them uniformly.

**Example**

* *Input*: Hive DESCRIBE FORMATTED output:

pgsql

CopyEdit

# col\_name data\_type comment

customer\_id string customer identifier

txn\_amount decimal(10,2) transaction amount

* *Parsing*: Transform into structured JSON:

json

CopyEdit

[

{ "column\_name": "customer\_id", "data\_type": "string", "comment": "customer identifier" },

{ "column\_name": "txn\_amount", "data\_type": "decimal(10,2)", "comment": "transaction amount" }

]

* *Standardization*: Convert types to canonical names (e.g., string -> STRING, decimal(10,2) -> DECIMAL).

**2. Semantic Tagger:** Analyzes column names, data types, and comments to **assign semantic tags** (like PII, Sensitive, Monetary). It Enables governance and automatic classification and assigns semantic meaning to technical metadata by tagging columns based on what they represent—e.g., customer names, account numbers, or sensitive data like PAN or Aadhaar.

**It** Uses pattern matching, column naming conventions, NLP models, or machine learning to infer semantics. It applies labels such as PII, customer\_identifier, geolocation, financial\_amount.

**Example**:  
If a column is named customer\_id, it is tagged as customer\_identifier, and account\_balance might be tagged as financial\_metric. A column named ssn could be tagged with PII.

* *Input Column*: customer\_id
* *Logic*: Regex matching + ML classification
* *Output Tag*: PII\_ID
* *Input Column*: txn\_amount
* *Output Tag*: MONETARY\_AMOUNT

**Quality Profiler :** The profiler assesses the quality of the data associated with each metadata entity by evaluating patterns, completeness, and consistency. It computes statistics such as null ratios, uniqueness, and distribution metrics. After parsing metadata, the profiler queries sampled or full data to calculate quality KPIs: % nulls, number of distinct values, outliers, and type mismatches. Profiles data values to compute **quality metrics** (nulls, uniqueness, min/max). It helps data stewards assess data health.

**Example**

For txn\_amount:

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Null % | 1.3% |
| Distinct % | 97% |
| Min | 0.00 |
| Max | 50000.00 |

**4. LoB–DB Mapper:**

In large organizations, the same database can serve multiple business domains. Maps databases to Lines of Business (LoBs), applications, and teams. This contextual mapping allows the catalog to group assets by business relevance. The system uses configuration rules, naming conventions (like prefix retail\_), or manual mappings to associate each database and schema with one or more LoBs.

**Example**

|  |  |  |
| --- | --- | --- |
| **Database Name** | **LoB** | **Application** |
| bank\_txn | Retail | Retail Transactions |
| bank\_txn | Wealth Mgmt | Portfolio Management |

**5. Type Mapper:**  Converts data types from source-specific formats to standardized canonical types used across the data catalog and processing systems. Each DBMS uses its own type system. The type mapper maintains a conversion table to map source types like VARCHAR2(255) (Oracle), STRING (Hive), or TEXT (Postgres) into a standard type like string.

**Example**

|  |  |
| --- | --- |
| **Source Type** | **Canonical Type** |
| varchar2(20) | STRING |
| bigint | LONG |
| decimal(10,2) | DECIMAL |

**6. Scheduler and Dependency Manager :** Schedules metadata extraction jobs and manages **dependencies** (e.g., extract lineage after schema extraction). It Ensures correct sequencing and retry handling. Manages the orchestration of metadata ingestion and ensures components run in correct sequence. Also maps interdependencies between tasks and components.

Implements job scheduling, retries, DAG management (similar to Airflow), and monitors job completion before triggering downstream jobs. It also identifies dependencies (e.g., table A must be profiled before lineage is built).

Once metadata is parsed from Hive, only then can profiling and semantic tagging jobs be triggered. Dependencies like “Profiling → Tagging → Lineage” are enforced.

**Example**

* *Job Schedule*:
  + **Schema Extraction**: Every night at 1AM
  + **Lineage Extraction**: After schema job succeeds
* *Dependency*: Lineage job depends on schema job completion.

**7. Business Metadata :**  Captures domain-level context such as business definitions, KPIs, classifications, owners, and LoB mappings. Unlike technical metadata, this is human-annotated and LoB-specific. Stored in a catalog database or metadata repository, often editable via GUI. Can be imported from business glossaries or entered by data stewards.

**Example**

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| Business Owner | John Doe |
| Purpose | Transaction Reporting |
| Retention Policy | 7 years |
| Confidentiality | Restricted |

**8. Operational Metadata:**  Describes metadata about the operations—such as data freshness, last ingestion timestamp, job status, version history, and access patterns. Captured by tracking job execution logs, timestamps, and schema change logs. Versioning systems can track “metadata about metadata” changes over time.

**Example**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Version** | **Extracted At** | **Change Summary** |
| txn\_table | v1 | 2024-05-01 01:00 | Initial load |
| txn\_table | v2 | 2024-06-01 01:00 | Added column currency |

**9. Lineage Metadata:** Stores **transformation relationships** between tables and columns. It Allows impact analysis and visualization. Captures upstream and downstream relationships between datasets—what data flows from what, including joins, views, transformations, and business processes. Parsed from SQL logs, ETL definitions, or view definitions. Often stored as a graph, showing parent-child relationships at table and column level.

**Example**

|  |  |  |
| --- | --- | --- |
| **Source Table** | **Target Table** | **Transformation** |
| raw\_txns | cleaned\_txns | Filter invalid records |
| cleaned\_txns | agg\_txns | Aggregate by date |

**10. Schema Metadata :** The foundational layer of technical metadata—includes table names, column names, data types, constraints, indexes, and comments. Stores canonical representation of tables, columns, and constraints. Foundation of all other metadata.

Extracted directly from system catalogs (information\_schema, dba\_tables, SHOW CREATE TABLE, etc.) and standardized via the parser.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | **Column Name** | **Data Type** | **Nullable** |
| txn\_table | txn\_id | STRING | No |
| txn\_table | amount | DECIMAL | Yes |

**11. Access Control :** Defines **who can view or edit** specific metadata. It Enforces governance and compliance. Defines who can view, edit, or administer which parts of the metadata or data. Implements RBAC (Role-Based Access Control). Each metadata object (table, column, glossary term) is tagged with access roles (e.g., Viewer, Steward, Owner). The catalog or API checks access rights before displaying or modifying metadata.

|  |  |
| --- | --- |
| **Role** | **Access Level** |
| Data Steward | Read/Write |
| Analyst | Read Only |
| Auditor | Read Only |

**12. Audit Logging:** Logs **who accessed or changed metadata**, when, and what was changed. It Supports compliance and traceability. Tracks all interactions with metadata: who changed what, when, and why. Crucial for compliance, debugging, and operational visibility. Each API/UI interaction or job output logs a metadata mutation or access request. Logs are stored with timestamps, IPs, and user identities.

**Example**

|  |  |  |  |
| --- | --- | --- | --- |
| **User** | **Action** | **Timestamp** | **Object** |
| jdoe | View | 2024-05-01 09:00 | txn\_table schema |
| admin | Edit | 2024-05-02 11:00 | Business Metadata |

**13. Lineage Logic :** The intelligent engine that parses transformation logic to extract lineage from SQL, ETL code, or views. Goes beyond table-level lineage to infer column-level flows. Derives lineage relationships by parsing **query logs and ETL job definitions**. It Automates lineage extraction. Uses SQL parsers, query plan analyzers, and string-matching of aliases and expressions to reconstruct how columns are derived across joins, filters, unions.

**Example**

*Parsed SQL:*

sql

CopyEdit

INSERT INTO agg\_txns

SELECT customer\_id, SUM(amount)

FROM cleaned\_txns

GROUP BY customer\_id;

*Lineage:*

* cleaned\_txns.amount → agg\_txns.SUM(amount)

**14. Impact Analysis:** Analyzes downstream consequences of changes to schemas, columns, or data logic. Helps assess risk before implementing changes.

Uses lineage graphs to find all dependent tables, views, reports, and users affected if a source changes.

Analyzes how changes propagate downstream. It Identifies what will break if a schema changes.

**Example**

*Change Detected:*

* Column txn\_amount dropped from cleaned\_txns

*Impact Report:*

* Breaks agg\_txns aggregation
* Affects 2 dashboards

**15. Filtering**: Applies filters to metadata for visualization and reporting. It Allows users to focus on relevant subsets. Allows users to slice and navigate metadata by dimensions like LoB, data domain, sensitivity, freshness, or tags. Implemented via search indexes, metadata tags, and UI facets. Allows dynamic filtering in dashboards or APIs.

**Example**

*Filter:*

* LoB = Retail
* Tables with PII columns

*Result:*

* Only Retail tables with PII data are shown in the ER diagram.

