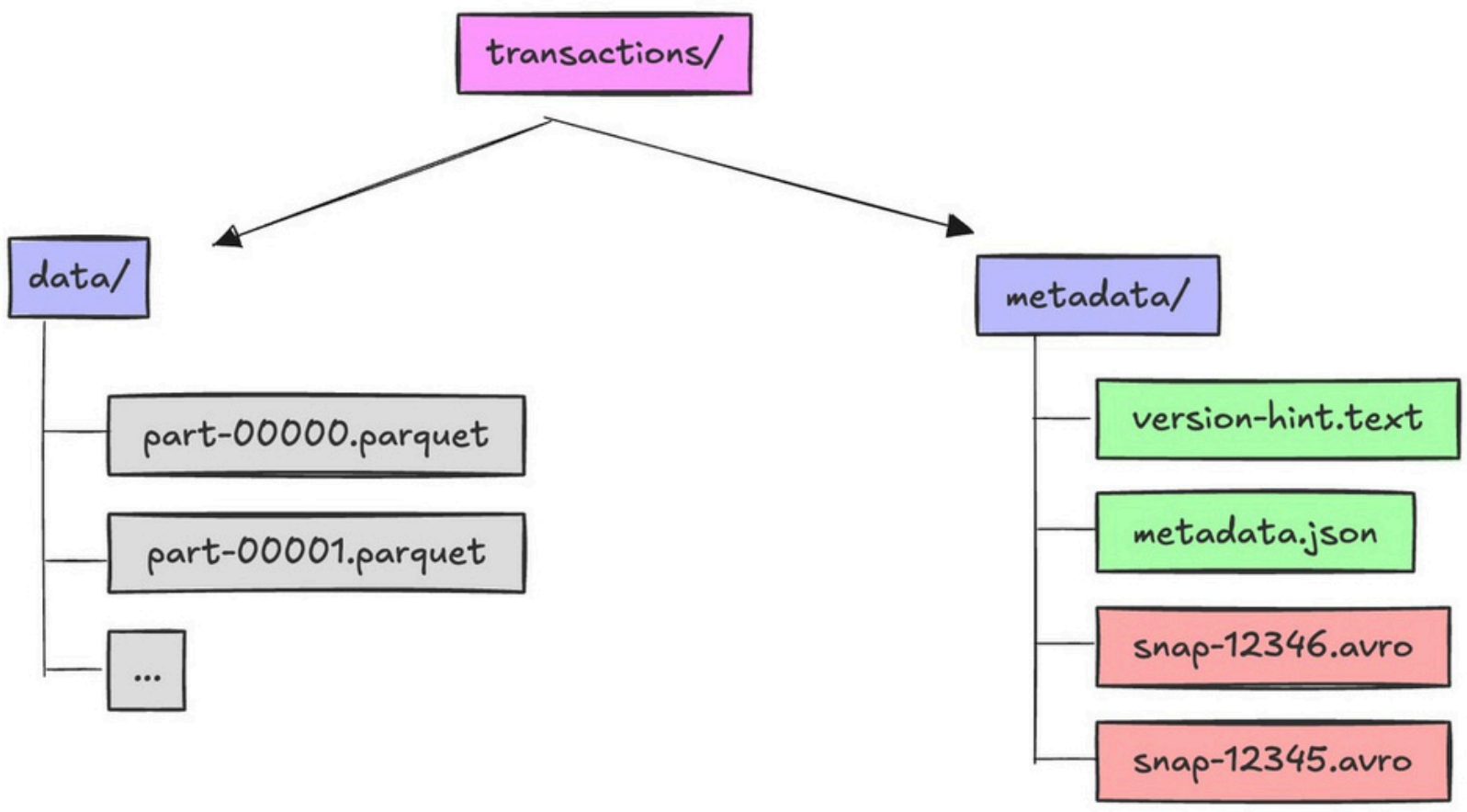
Why do you need an Iceberg Catalog?

An Iceberg catalog is a centralized service that manages and tracks Iceberg tables, providing a unified interface for table discovery, access control, and metadata management across different compute engines.
Although metadata lives alongside the Iceberg table data, the catalog is essential for discoverability, safe concurrent operations, logical naming, and multi-engine compatibility.

In this sketch note, we'll explore what an Iceberg catalog is, why we need one, and how its architecture works.

Iceberg Metadata Structure

If you crack open an Iceberg table, you will see that its metadata is stored alongside the actual table data.
For example, the ***transactions*** table will have an on-disk storage layout like this:



In this layout:

- The ***data/*** directory contains the actual data files in Parquet format
- The ***metadata/*** directory stores:
- Snapshot files (.avro) that track table modifications
- ***metadata.json***, which maintains the table's current state
- ***version-hint.text*** pointing to the current metadata version

Follow Me @dunithd /in/dunithd/

Why do you need an Iceberg Catalog?

Every table modification (DDL/DML operation) creates an isolated snapshot in Iceberg. These snapshots are tracked in the **metadata.json** file, which maintains a complete history of the table's state over time. The most recent version of **metadata.json** serves as the source of truth for the current table state, including schema, partitioning, and data file locations. The name of this metadata file is stored in a file with a typical name like ***version-hint.text*** file.

Using Iceberg table - without a catalog

An Iceberg catalog is a centralized service that manages and tracks Iceberg tables, providing a unified interface for table discovery, access control, and metadata management across different compute engines.

Now, imagine you're a data engineer or analyst working with over 100 Iceberg tables daily. What challenges would you face without an Iceberg catalog?

Discovery and Management - There's no central registry of what tables exist, their namespaces, or how to reference them.

- You must know the full storage path (like ***s3://my-bucket/data/warehouse/sales.db/transactions/***) to interact with a table.
- **Development challenges:** Applications would need hardcoded storage paths. Moving them between dev/test/prod would require path changes
- T**able enumeration:** No way to programmatically list or discover existing tables
- **Namespace organization:** Loss of logical grouping - tables would exist as isolated entities

Concurrent Access Issues - Multiple writers could create conflicting commits without coordination

- **Transaction conflicts:** No mechanism to prevent overlapping operations that could corrupt the table state
- **Lock management:** No way to coordinate exclusive operations like schema changes

Why do you need an Iceberg Catalog?

Integration Complexity - Each engine would need direct file system access and custom logic to locate tables

- **Tool compatibility:** Many data tools expect catalog APIs and wouldn't work without them
- **Metadata caching:** No shared caching layer, leading to repeated expensive metadata reads

Security and Access Control - You'd need to rely solely on file system permissions

- **Audit challenges:** Difficult to track who accessed or modified which tables
- **Authentication gaps:** No single point for credential management

Key Reasons Why Iceberg Catalogs Exist

Here are the key reasons for using an Iceberg catalog.

Human-Readable Table Names and Namespaces

Users and query engines work with logical names like ***sales.transactions*** instead of storage URIs. When you run a SQL query, the catalog translates these names into the underlying metadata and storage paths.

```
SELECT * FROM sales.transactions;
```

Resolved by the catalog:

```
s3://my-bucket/warehouse/sales/transactions/metadata/metadata.json
```

Why do you need an Iceberg Catalog?

Multi-engine Interoperability

Catalogs provide a common lookup interface for multiple compute engines. Without it, engines would have to reimplement logic to find and interpret table locations and versions.

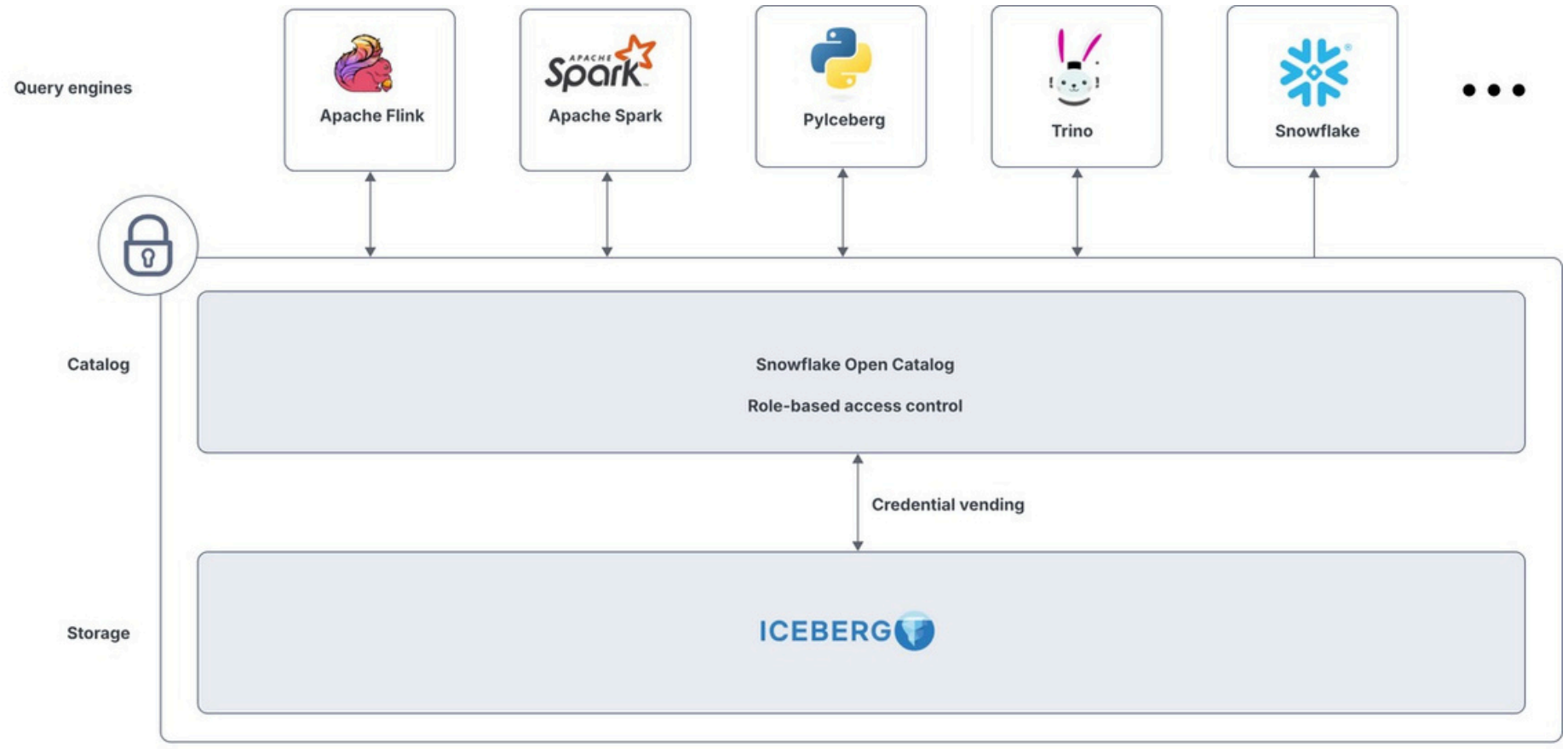
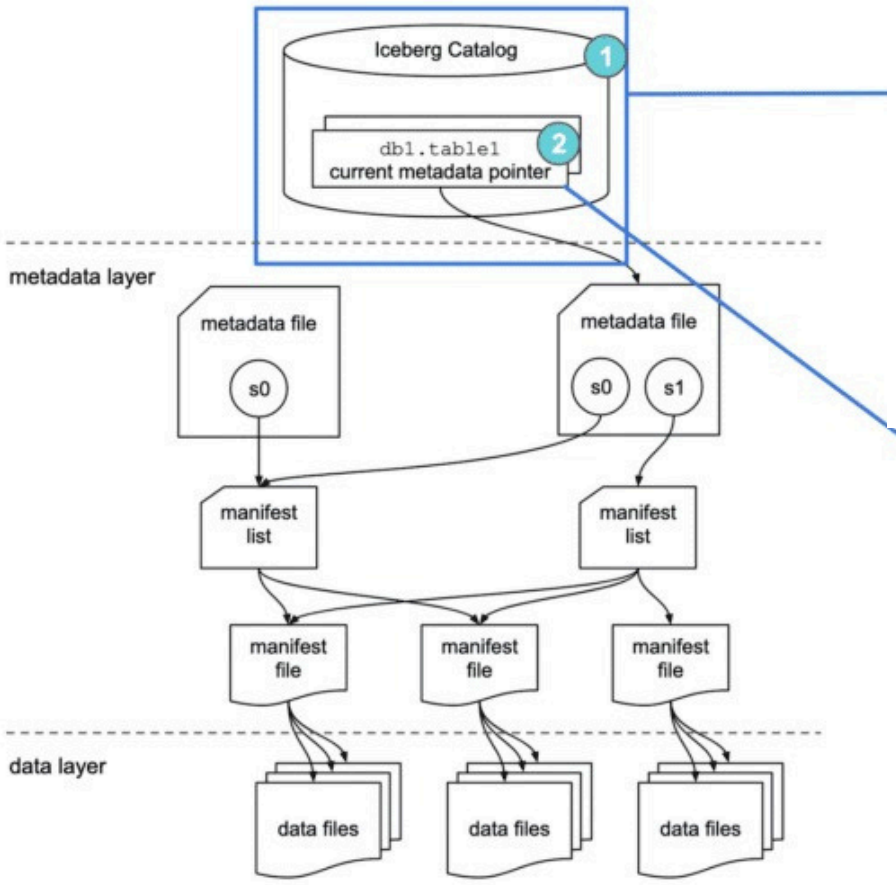


Image credits - https://other-docs.snowflake.com/en/opencatalog/overview

Snapshot Management



Catalogs maintain the reference to the current **metadata pointer** (latest snapshot), enabling time travel, atomic schema evolution, and safe concurrent writes across engines.

`table1`'s current metadata pointer

- Mapping of table name to the location of current metadata file

Why do you need an Iceberg Catalog?

Versioned Table References

Some catalogs (like Project Nessie) offer branching and tagging, providing Git-like version control for tables. These catalogs can maintain multiple versions of the same table (e.g., ***main***, ***dev***, ***v1.0***).

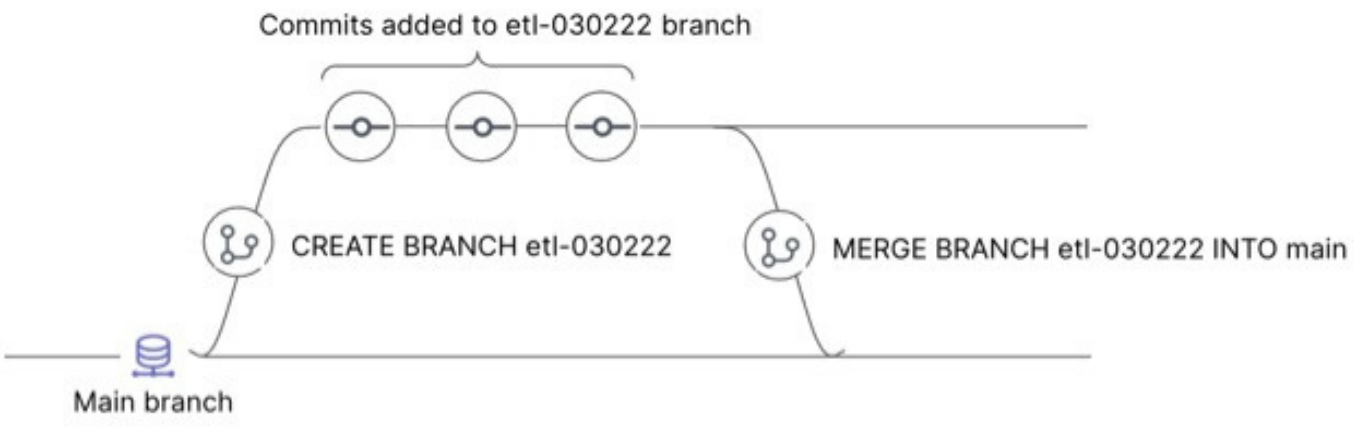


Image credits - https://docs.dremio.com/24.3.x/sonar/query-manage/managing-data/nessie/

Schema Evolution and Table Evolution

Catalogs track changes in schema, partitioning, and properties, exposing them cleanly to engines. Since metadata files update frequently, the catalog ensures the current version is always referenced correctly.

Atomicity and Consistency

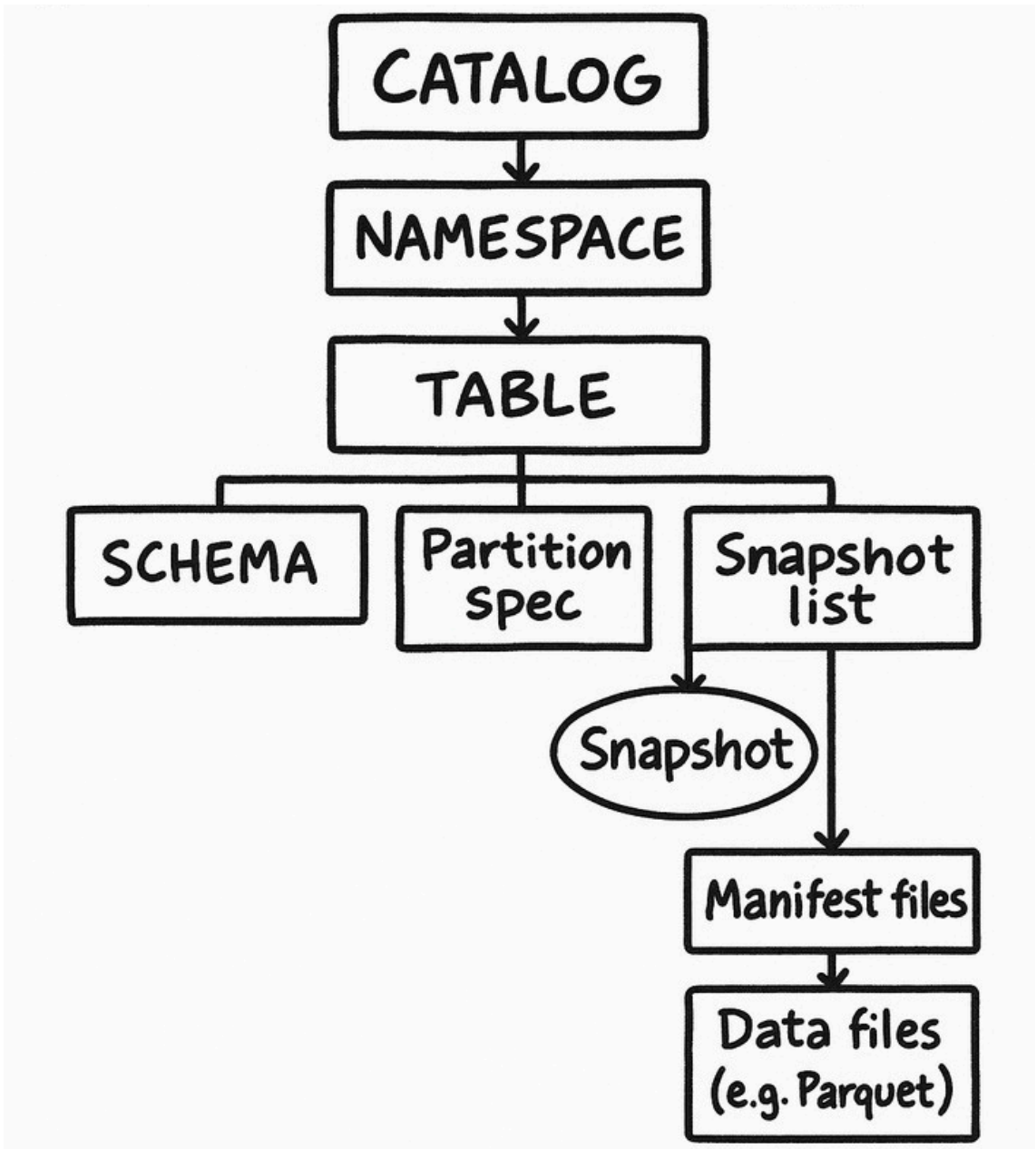
A catalog ensures atomic table operations, like replacing a snapshot, remain consistent even when multiple users or engines are accessing the table simultaneously.

Why do you need an Iceberg Catalog?

Catalog Architecture

Now that we understand why catalogs are important, let's look at how they actually work under the hood. Breaking down a catalog's architecture helps us see how it manages all those tables and keeps everything running smoothly.

At a high level, a catalog is made of three entities: ***catalog***, ***namespace***, and ***table***.



Catalog - Top-level container in the hierarchy

- Manages connection to underlying storage and metadata repository
- Examples: AWS Glue Catalog, Hive Metastore, JDBC catalogs, custom REST catalogs

Responsibilities:

- Tracks all tables across an organization
- Provides authentication and access control
- Maintains namespace organization
- Serves as a transaction coordinator

Why do you need an Iceberg Catalog?

Namespace - Middle-level organizer (similar to databases or schemas in traditional systems)

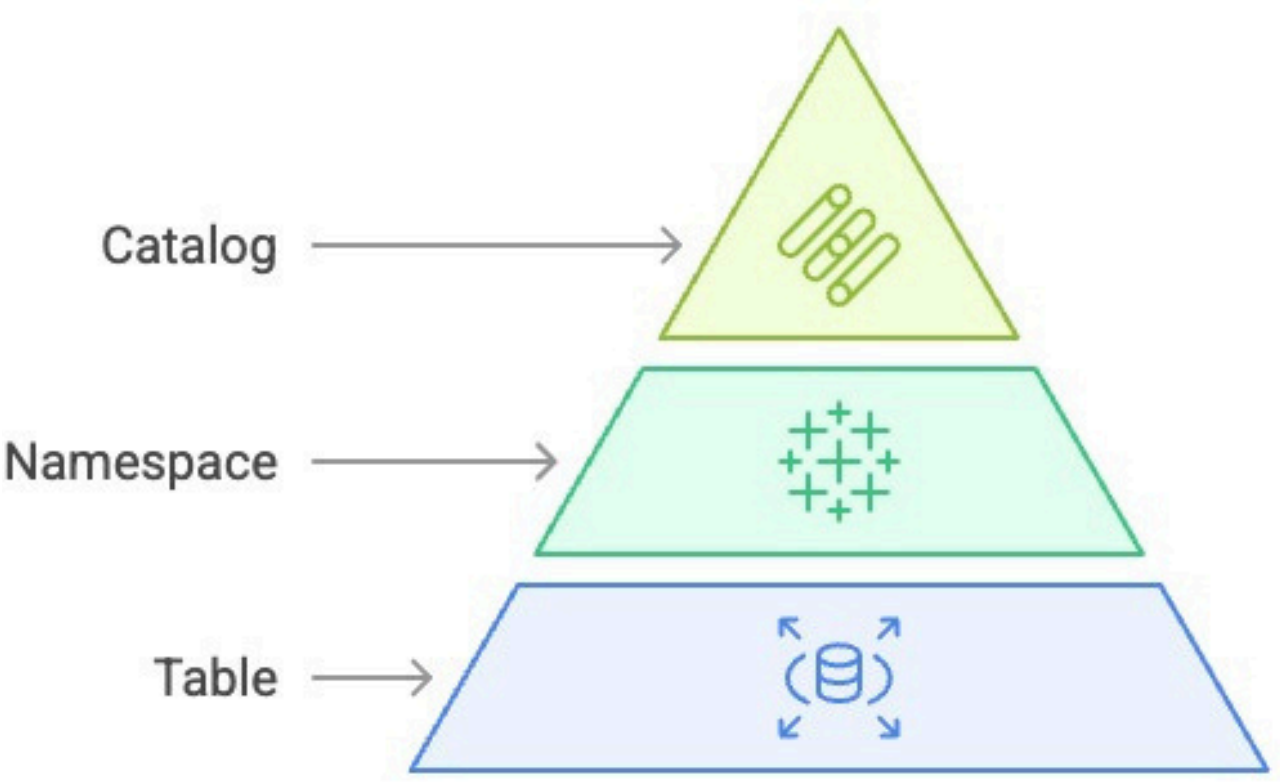
- Lives within a catalog
- Groups related tables logically
- Examples: "production", "finance_dept", "ml_features"
- Typically represented as a path-like structure (e.g., "db.schema" or "tenant.database")

Table - Lowest-level entity containing the actual data

Lives within a namespace and contains:

- Data files (Parquet, ORC, Avro)
- Metadata files (schema, partitioning, snapshots)
- Manifests listing data files

You can use the following visual to easily understand the component hierarchy.



Why do you need an Iceberg Catalog?

Iceberg Catalog Implementations

Several popular Iceberg catalog implementations offer distinct features and use cases. **AWS Glue Catalog** provides native integration with AWS services and seamless compatibility with Amazon EMR and Athena. **Apache Hive Metastore**, being one of the earliest implementations, offers broad compatibility with various Hadoop ecosystem tools. **Project Nessie** stands out by providing Git-like versioning capabilities, enabling branch-based development and experimentation. **REST Catalog** implementations allow for custom, scalable solutions that can be integrated with existing infrastructure.

Each catalog implementation has its strengths - AWS Glue excels in AWS environments, Hive Metastore provides traditional reliability, and Nessie offers advanced versioning features for modern data workflows.

Why do you need an Iceberg Catalog?

Wrapping it up

Think of an Iceberg catalog like the **pg_catalog** in Postgres or the **information_schema** in MySQL — it's the registry of what exists, how to reference it, and what its latest structure is.

Although metadata files live alongside the data, using Iceberg tables without a catalog is impractical for real-world applications beyond simple experiments. Without a catalog, Iceberg becomes a lower-level abstraction — while still powerful, it's much harder to use safely in production environments with multiple tools.

The catalog is essential for discoverability, safe concurrent operations, logical naming, and multi-engine compatibility.