



WHITEPAPER

Data Warehousing Guidelines with Coalesce

Best Practices for Planning, Building, and
Operating Your Snowflake Data Pipelines

by Douglas Barrett, Principal Solutions Architect, Coalesce

Table of Contents

01 Introduction

Data Warehouse Layers in Detail

04 Building With Coalesce

04 Project Approach

05 Preparation

Naming Standards When Building with Coalesce

06 Development

Source

Data Persistence

Data Rules

Detail Star Preparation

Detail Star Schema

Aggregate Star Schema – Preparation

Aggregate Star Schema

Semantic Layer

10 Deployment

11 Operation

12 Conclusion

13 About Coalesce

Introduction

This is a high level guide to planning, building, and operating a data warehouse on Snowflake using Coalesce. This document suggests standards and guidelines that can be used in full or in part.

Coalesce allows data engineers and analysts to rapidly build data processing pipelines – specifically designed for data warehouse or analytic processing – using SQL on Snowflake. Coalesce automates the generation of SQL using patterns from editable templates and pushes it down to Snowflake.

Generically, the data warehouse is built in standardized layers to simplify build and management. These layers could be all or a subset of:



Note: Business Rules are usually maintained in the preparation area of the star schema layers. Data Rules and Business Rules might be separated from a logical standpoint or combined from a practical (performance) standpoint.

Data Warehouse Layers in Detail

- **Source** refers to raw data that is accessible using Snowflake SQL – either accessing data already ingested into Snowflake using an ingest tool such as Fivetran or HVR, or data that is accessible in cloud storage or a data lake using Snowflake External Tables or COPY command (which Coalesce can build and manage).
- **Data Persistence** is a historical or time variant record of source data. This should be a reliable representation of raw data and should be able to reconstruct the data as it was received. A persistent layer might be made up of a Data Store or Data Vault, or a combination of the two. In more detail:
 - *Data Store*: persistent tables populated using either an INSERT (WHERE NOT EXISTS) or a MERGE DML statement using a business key. Only the most recent version of data is persisted.
 - *Historical Data Store*: persistent tables in which historical or time variant records of changes are detected and managed using a Start and End date – similar to a Type 2 Slowly Changing Dimension.
 - *Data Vault*: a standardized and flexible modeling approach for persisting data – dividing data into keys (Hubs), attributes (Satellites) and relationships (Links). Data Vault is particularly powerful when managing data from multiple source systems and changing source landscapes.
- **Data Rules** are implemented using SQL in a staging table or view to apply hard data rules such as:
 - NULL handling
 - Data type conversion
 - De-duplication / referential checks and flagging
 - Quality checks / flagging
 - Incremental filtering – to select only new or changed data each run.
There is value in creating a stage view over each individual source node to keep the pipelines consistent, even if it is not required for each source node.

There is value in creating a stage view over each individual source node to keep the pipelines consistent, even if it is not required for each source node.

- **Detail Star Preparation** applies processing to prepare data for publishing into the detail layer of the star schema layer. Tables in this and the detailed star layer are at the grain of the raw data. Transformations include business rules such as:
 - Denormalization – particularly for dimensional data
 - Standardized metric calculations (e.g. Revenue, Margin, etc.)
 - Decoding (e.g. M -> Male, F -> Female, etc.)
 - Filtering based on business use (e.g. exclude returns from sales data)
 - Business-friendly attribute names and descriptions
 - Dimension key lookups

Testing should be added to this layer of processing to ensure that the data meets the quality needed to publish to the star layers.

- **Detail Star:** star schema layer at the raw grain of data – typically used as a common base to dimensionalize and support downstream aggregations. This layer is seldom directly accessed by users but is used to support drill-down reporting. Data is modeled into dimensions (combination of type 1, type 2 and role playing) and Facts are typically in a single subject area at transaction level grain or snapshot (e.g. account balance per day).t

The star schema will consist of shared / conformed dimensions that span multiple subject areas, together with subject area-specific dimensions and facts.

- **Aggregate Star Preparation** applies processing to prepare data for publishing into the aggregate layer of the star schema layer. This processing includes:
 - *Aggregation* – based on dimensional hierarchies (e.g. monthly / region)
 - *Integration* – join facts on common level of aggregation / dimension keys
 - *KPI Metrics* – metrics that are defined on aggregate data (e.g. year on year comparison)

Testing should be added to this layer of processing to ensure that the data meets the quality needed to publish to the aggregate star layer.

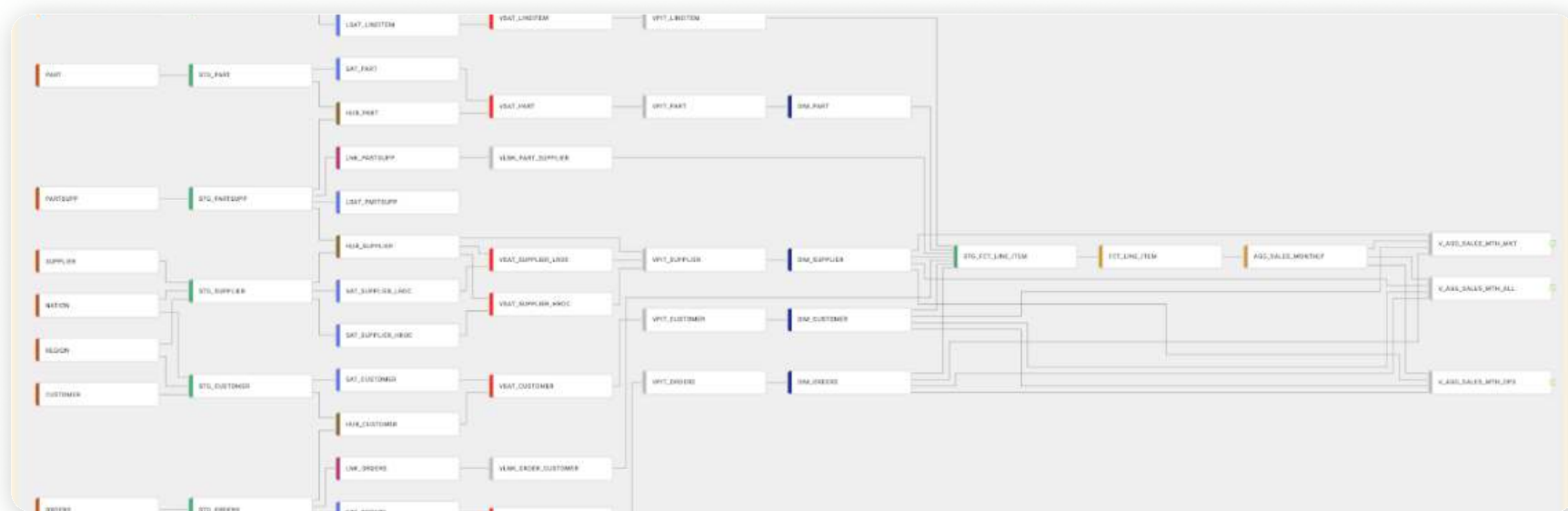
- **Aggregate Star Schema:** data is aggregated and combined across subject areas (e.g. sales and inventory). Most reports / dashboards query this aggregated view of data (e.g. Sales by Store / Brand or Claims by Provider, etc.). Data is aggregated from the detail star schemas and, if necessary, combined by common dimensional keys or hierarchies.
- **Semantic Layer:** a layer of views that can apply the last mile naming, simplifying, and access control. Semantic views can also add calculations (be wary of slowing them down), add joins to flatten the presented data structure, business naming standards, masking, and filtering rules. When passthrough connections identify a user or role, the data warehouse can leverage Snowflake's filtering and masking based on the report / query user role.

Ready to try what you've learned so far with Snowflake and Coalesce?

[START YOUR FREE TRIAL NOW](#)

Building With Coalesce

Coalesce can build the structures and processing required to take raw data ingested from a data source through a pipeline of processing and persistence into a consumable semantic layer. An example of this processing in Coalesce looks like the image below, showing pipelines from source, through persistence, detail star, aggregate star, and semantic views:



End-to-End Pipeline view in Coalesce

Project Approach

Coalesce supports rapid development and deployments with flexibility and standards built in. An agile development approach is supported – but so, too, is a more traditional waterfall style project if that fits the culture of the organization.

Typically, the data warehouse project needs to be agile and move quickly to maintain engagement with Subject Matter Experts (SMEs). It is important to build through layers rapidly, from source to star schema, to prove the data quality, referential quality, feasibility, and data possibilities.

Based on a high level design, a prototype star schema should be built and socialized with SMEs – dragging through many of the source attributes with their original column names and data types. Production source data should be used where possible. Development shortcuts can be made building the Prototype in the interests of speed, but must be revisited during polishing into a production-ready system.

Note: expectations should be managed with SMEs. The Prototype is not meant to be a polished system; it is meant to socialize data characteristics and design feasibility. Polishing the prototype into a production-ready system is a project phase that takes time (a separate sprint), and the project feasibility / possibilities should be explored before spending time polishing.

The persistent layer and detail fact can largely be built based on source data structures. Tests, renaming, pruning, and transforms can be added based on feedback from the SMEs working with the Prototype environment.

Once the core objects and attributes have been proven in the Proof of Concept, then the POC can be polished into a production system by applying appropriate naming, tests, standards, pruning (of unwanted objects / attributes), and adding aggregations, calculations and views.

Preparation

Each layer of processing should have its own storage location (mapping to a database / schema), its own naming pattern, and its own processing standards. In more detail:

- **Storage Locations:** Each layer of the data warehouse should have at least one mapping in Coalesce, which maps to a physical database and schema in Snowflake for each environment. There may be multiple mappings for source data that map to database / schemas containing raw data ingested into Snowflake. Also, the semantic layer views will exist in either a Shared or Business specific mapping.
- **Templates:** Allow a team to modify the default naming of a node, the default storage location of a node, and the code patterns to build and populate a node.
- **GIT integration:** Necessary to commit changes into version control. Coalesce's ability to leverage a GIT repository can be used to support team development in feature branches and development workspaces. Deployments are made from a GIT branch into downstream environments, either using the Web UI or a command line script.
- **Naming standards / storage defaults:** Ensure that these standards exist and that the developers know about them. Retrospectively changing either can take a while.

Naming Standards When Building with Coalesce

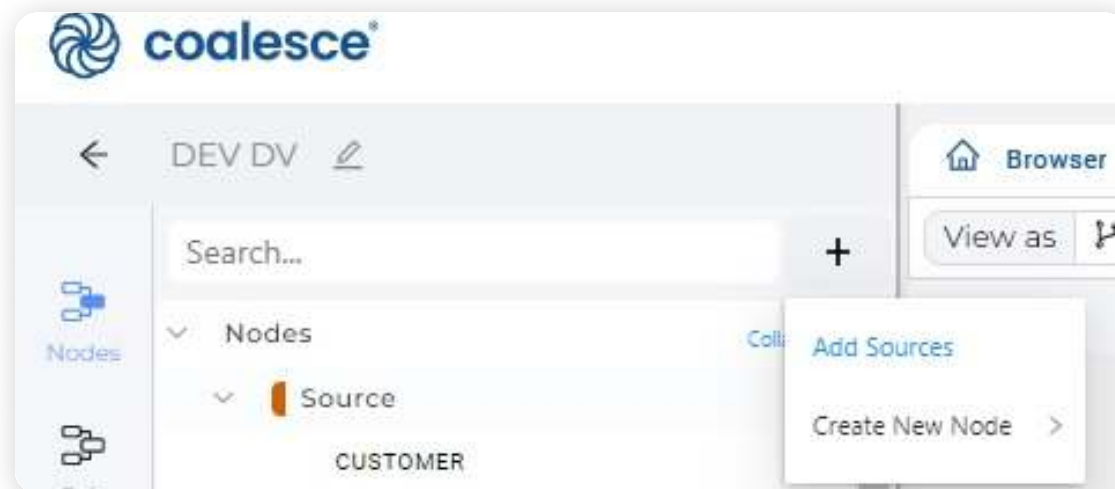
When building with Coalesce, each Node type can have its own naming pattern / standard applied automatically, e.g. STG_ as a prefix for Stage Nodes. This is a useful start of a naming standard. A thoughtful naming standard that is consistently applied by developers has a huge and enduring value as the data warehouse grows. Naming standards can be used to apply User Interface filters to find specific nodes that are related to a source, subject or object. These names can also be used to define a processing strategy (job definition) such as process data vault nodes from source A, source B, then process conformed dimensions, then subject area 1, area 2 – using that naming standard.

Development

Once you've done the preparations, you're ready to start building your data warehouse. In more detail:

Source

Once mappings are defined to source data, the Source nodes can be imported into Coalesce using the **Add Sources** option on the + button in the **Nodes** menu:



Adding source data in Coalesce

Once Source nodes have been selected, then we can start building a pipeline.

Data Persistence

This layer provides a foundation of raw data from which all downstream nodes and processing can be built or rebuilt.

There are three main approaches to this layer of the data warehouse:

- The source or ingest tool maintains raw data at the right grain forever (e.g. a data lake or replicated layer). This means that no separate persistence layer is needed in Coalesce – move to the next layer.
- A data store is a simple persistence layer, typically 1 for 1 per source table, built using a Persistent Stage (or variant of Persistent Stage) that either stores the latest version of data or a historical / time variant version of source data. To build this layer a standardized naming can applied, such as:

`“SOURCE TABLENAME” -> “NODE TYPE”_”SHORT SOURCE NAME”_”SOURCE TABLE_NAME”`

eg

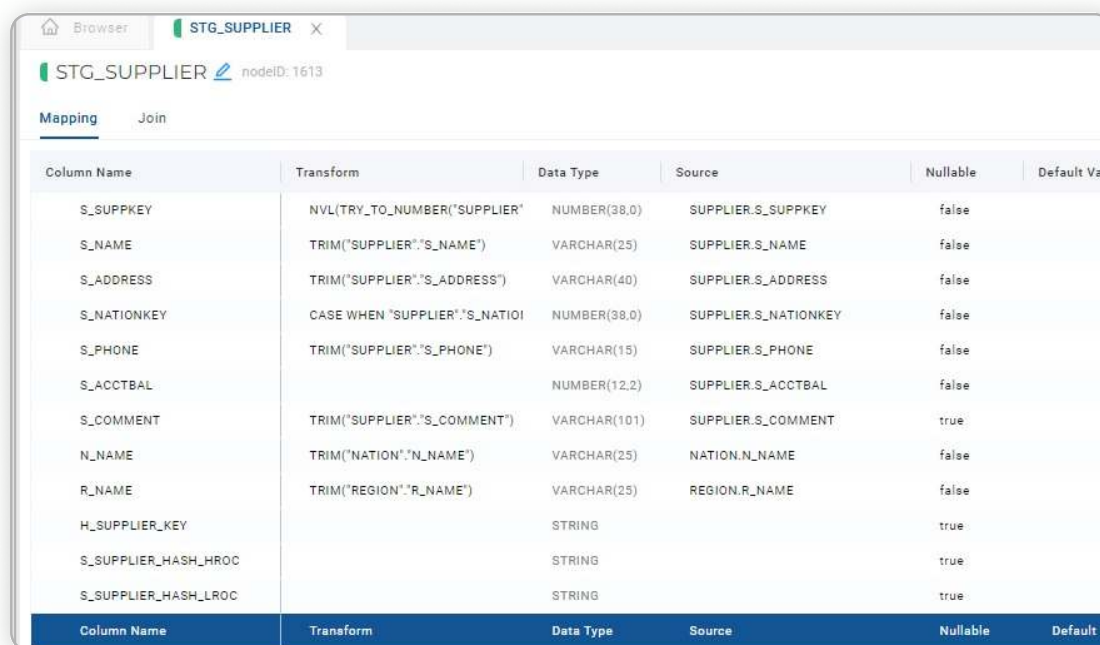
`SUPPLIER -> PSTG_OPS_SUPPLIER`

- A data vault is a more complex and sophisticated persistence layer that standardizes data around keys (hubs), attributes (satellites) and relationships (links). Coalesce has a package of templates developed specifically for building and managing a data vault layer (if these do not exist in your environment please contact Coalesce support).

Data Rules

This layer of the data warehouse prepares data for the star schema layers by converting data types, replacing NULLs, deduplicating and filtering the data that is useful for user consumption.

Typically built 1 for 1 with each data persistence table, these will be stage tables or views. In Coalesce, these nodes are built using non-persistent stage tables or views, which apply transforms to ensure the data and data type is correct. For example:



Column Name	Transform	Data Type	Source	Nullable	Default Value
S_SUPPKEY	NVL(TRY_TO_NUMBER('SUPPLIER',	NUMBER(38,0)	SUPPLIER.S_SUPPKEY	false	
S_NAME	TRIM('SUPPLIER'.S_NAME')	VARCHAR(25)	SUPPLIER.S_NAME	false	
S_ADDRESS	TRIM('SUPPLIER'.S_ADDRESS')	VARCHAR(40)	SUPPLIER.S_ADDRESS	false	
S_NATIONKEY	CASE WHEN 'SUPPLIER'.S_NATIOI	NUMBER(38,0)	SUPPLIER.S_NATIONKEY	false	
S_PHONE	TRIM('SUPPLIER'.S_PHONE')	VARCHAR(15)	SUPPLIER.S_PHONE	false	
S_ACCTBAL		NUMBER(12,2)	SUPPLIER.S_ACCTBAL	false	
S_COMMENT	TRIM('SUPPLIER'.S_COMMENT')	VARCHAR(101)	SUPPLIER.S_COMMENT	true	
N_NAME	TRIM('NATION'.N_NAME')	VARCHAR(25)	NATION.N_NAME	false	
R_NAME	TRIM('REGION'.R_NAME')	VARCHAR(25)	REGION.R_NAME	false	
H_SUPPLIER_KEY		STRING		true	
S_SUPPLIER_HASH_HROC		STRING		true	
S_SUPPLIER_HASH_LROC		STRING		true	

Example data transforms in Coalesce

Note: When converting data, we typically use a TRY_TO function in Snowflake to limit failures.

As this is the start of a processing pipeline, it is often a good place to apply incremental processing filters so that only new or changed data is moved through the pipeline. An example in Coalesce is defined in the JOIN definition referencing a downstream persistent table. In this example, the **ref_no_link** function in Coalesce is used to reference a downstream persistent table without breaking the one-way nature of DAG:

```
23 TRY_PARSE_JSON("SALES_ORDERS"."CLICKED_ITEMS") AS "CLICKED_ITEMS",
24 TRY_PARSE_JSON("SALES_ORDERS"."PROMO_INFO") AS "PROMO_INFO",
25 "SALES_ORDERS"."_FIVETRAN_DELETED" AS "_FIVETRAN_DELETED",
26 "SALES_ORDERS"."_FIVETRAN_SYNCED" AS "_FIVETRAN_SYNCED"
27 FROM
28 "FIVETRAN_DATABASE"."RETAIL_DATA"."SALES_ORDERS" "SALES_ORDERS"
29 WHERE
30 _FIVETRAN_SYNCED > (
31 SELECT
32 NVL(MAX(_FIVETRAN_SYNCED), '1900-01-01')
33 FROM
34 "DOUG_DB"."DEV_EDW_DOUG"."FCT_SALES_ORDERS"
35 )
```

Incremental Pipeline Filter example

Note: When joining tables together, it is often only the driving table that needs this incremental filtering applied. Smaller reference tables should be fully populated.

The naming convention for this processing layer should reflect the source:

“SOURCE TABLENAME” -> “NODE TYPE”_”SHORT SOURCE NAME”_”SOURCE TABLE_NAME”

eg

SUPPLIER -> STG_OPS_SUPPLIER

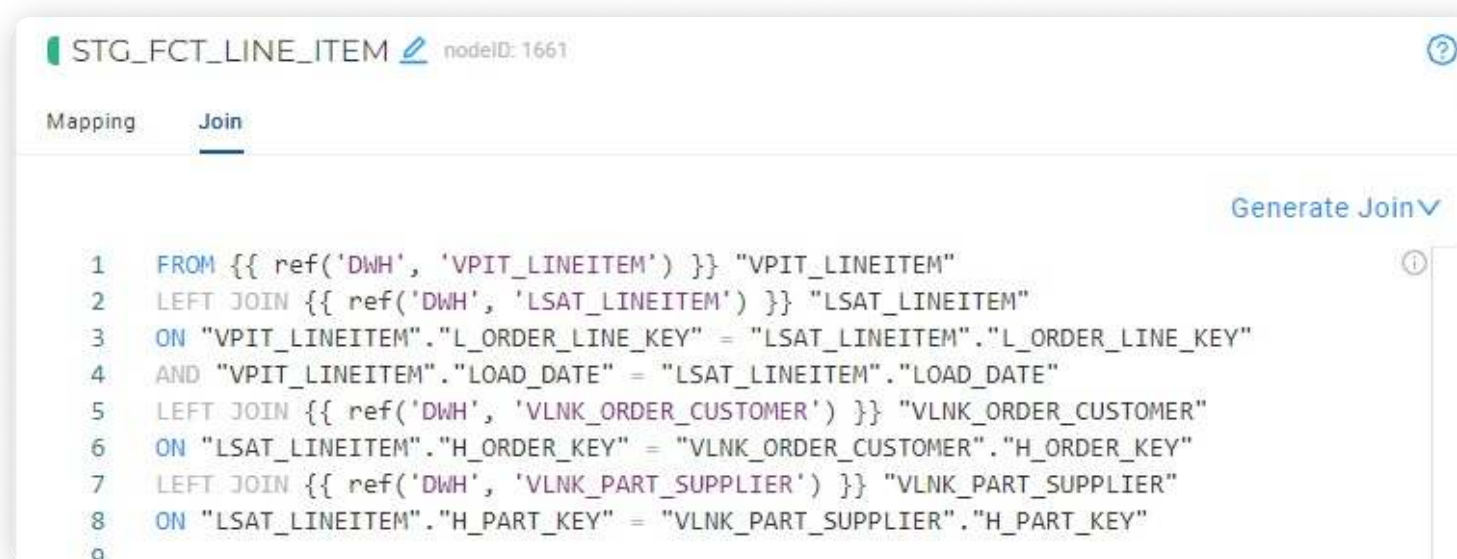
Detail Star Preparation

This layer of the data warehouse takes the cleaned, detailed data and prepares it for the detail star schema. The detail star schema is often at the lowest grain of the raw data and is used as the immediate source for the aggregate stars downstream.

The processing in this layer of processing will apply the transformations that can be applied at a detail grain and dimensionalize the data into facts and dimensions.

Coalesce has built-in templates that support the denormalization and change tracking of dimensional data, which automatically adds in support for surrogate keys and change tracking columns. It is very quick and easy to build a Type 1 or Type 2 Slowly Changing Dimension with this template.

The standard stage template can be used for preparing data for publish to a fact by LEFT JOINing to the dimensions on the business key of the dimensions to look up surrogate keys. For example:



```
STG_FCT_LINE_ITEM nodeID: 1661
Mapping Join
Generate Join v
1 FROM {{ ref('DWH', 'VPIT_LINEITEM') }} "VPIT_LINEITEM"
2 LEFT JOIN {{ ref('DWH', 'LSAT_LINEITEM') }} "LSAT_LINEITEM"
3 ON "VPIT_LINEITEM"."L_ORDER_LINE_KEY" = "LSAT_LINEITEM"."L_ORDER_LINE_KEY"
4 AND "VPIT_LINEITEM"."LOAD_DATE" = "LSAT_LINEITEM"."LOAD_DATE"
5 LEFT JOIN {{ ref('DWH', 'VLNK_ORDER_CUSTOMER') }} "VLNK_ORDER_CUSTOMER"
6 ON "LSAT_LINEITEM"."H_ORDER_KEY" = "VLNK_ORDER_CUSTOMER"."H_ORDER_KEY"
7 LEFT JOIN {{ ref('DWH', 'VLNK_PART_SUPPLIER') }} "VLNK_PART_SUPPLIER"
8 ON "LSAT_LINEITEM"."H_PART_KEY" = "VLNK_PART_SUPPLIER"."H_PART_KEY"
9
```

Join definition in Coalesce Node details

The naming convention for this processing layer should reflect the target:

“NODE TYPE”_”TARGET TABLE_NAME” -> “TARGET TABLE”

eg

STG_DIM_SUPPLIER -> DIM_SUPPLIER

Detail Star Schema

This layer is a dimensionalized representation of the key transactions and business objects in a simplified, persistent star schema. Though this data is seldom used by BI tools and reporting, it is available for drill down reporting. All aggregations are built from this detail layer to provide consistency at the different levels of aggregation required by the business. Tables are either Fact, Dimension or Bridge.

Facts are almost always subject specific and will include the name of the subject or business area in their names. Dimensions will either be subject specific or conformed across the organization – non-conformed dimensions will also include the subject in their names.

Subject specific tables naming:

“NODE TYPE”_”SUBJECT”_”OBJECT”

eg

DIM_SALES_BRANCH

FACT_SALES_INSTORE

Conformed Dimension naming:

“NODE TYPE”_”OBJECT”

eg

DIM_CUSTOMER

Aggregate Star Schema – Preparation

This layer takes data from the detail star layer and prepares it for the aggregate layer. The key transforms will be joining facts on common dimension keys, aggregations, and aggregate transformations. Intermediate work tables / views should be incremental and non-persistent.

The naming of this stage will again reflect the target tables:

“NODE TYPE”_”TARGET TABLE” -> “TARGET TABLE”

eg

“STG_AGG_SALES_MONTHLY” -> “AGG_SALES_MONTHLY”

Aggregate Star Schema

This layer typically consists of persistent tables modeled as dimensions, aggregate facts table, and / or flattened table structures, and is accessed via the Semantic layer views.

The naming of this stage will often contain the SUBJECT and level of aggregation:

“NODE TYPE”_”SUBJECT”_”AGGREGATE LAYER”

eg

“AGG_SALES_REGION_MONTHLY”

“AGG_SALES_DAILY”

“AGG_INVENTORY_DAILY”

Semantic Layer

This is a layer of views that are directly accessed by BI, reporting, and data science tools. These views offer a layer of abstraction to the underlying tables, offer a security framework by leveraging Snowflake’s Role Based Access Control over Views and Schemas, and masking using Snowflake’s masking policies. All of the DDL to build out this layer can be built using Coalesce.

Naming is based on the SUBJECT and optionally the target user group name:

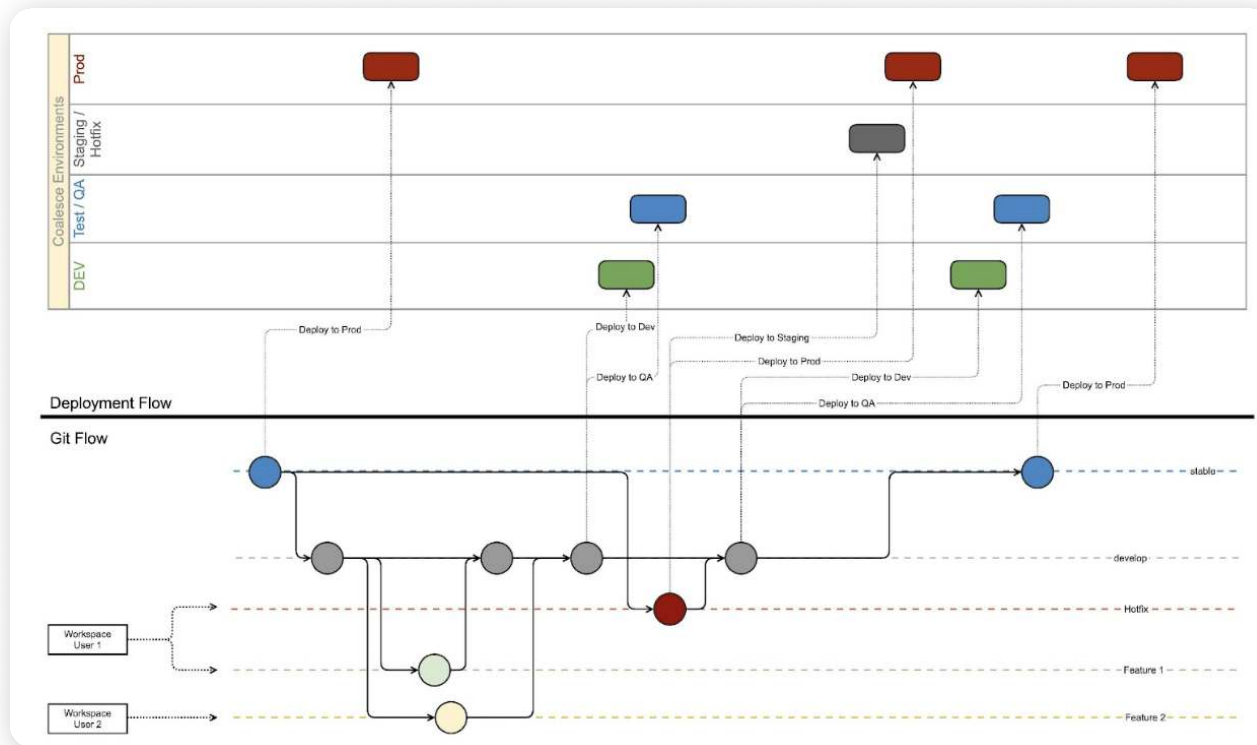
“NODE TYPE”_”SUBJECT”_”AGGREGATE”_”USER GROUP”

eg

“V_SALES_MONTHLY_MKT”

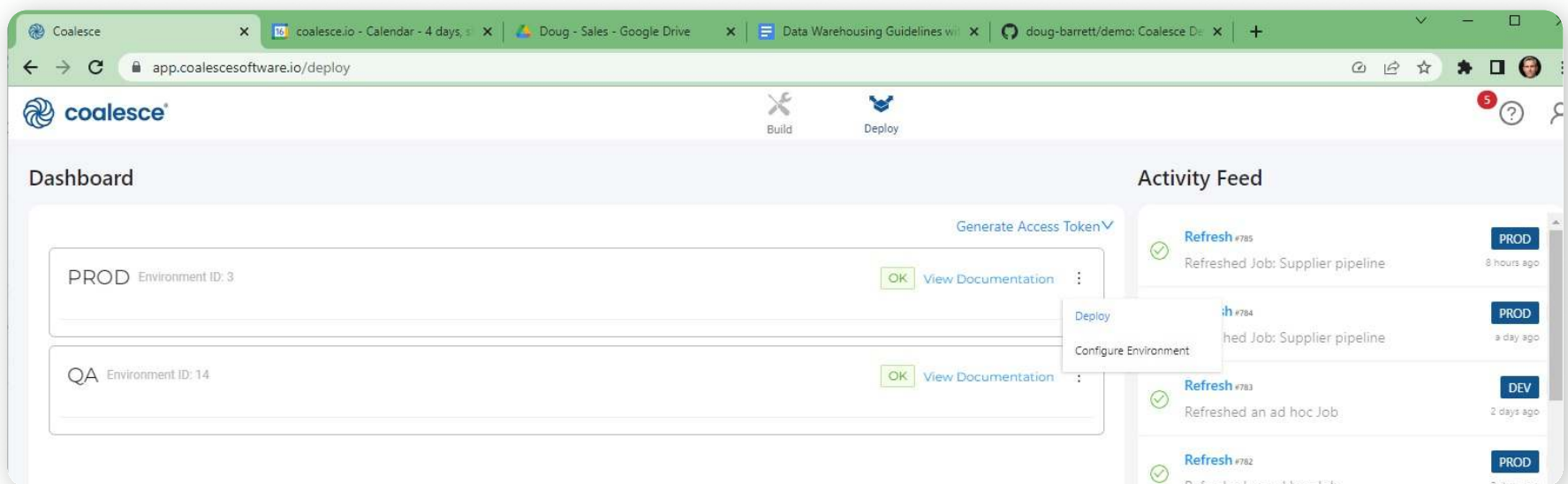
Deployment

Coalesce supports all of the versioning and branching capabilities of GIT – allowing the development team to use parallel development of feature branches in their own developer workspaces until they are ready to merge. Typically, a stable deployment branch is managed that contains the merges from unit-tested feature branches. This stable branch is protected in the GIT repository so only a Team Lead can merge.



GitLab Inspired GIT branching

Once a development has been tested and is ready for deployment into (pre-)Production, Coalesce can generate a plan from a chosen GIT Commit against previously deployed schema. This plan will show the generated DDL to synchronize Production schema with the chosen Commit. If this passes review, the Deployment can be run from Coalesce. Logs from the DDL execution will be written to Coalesce's Activity Log.



Deployment Interface in Coalesce

Operation

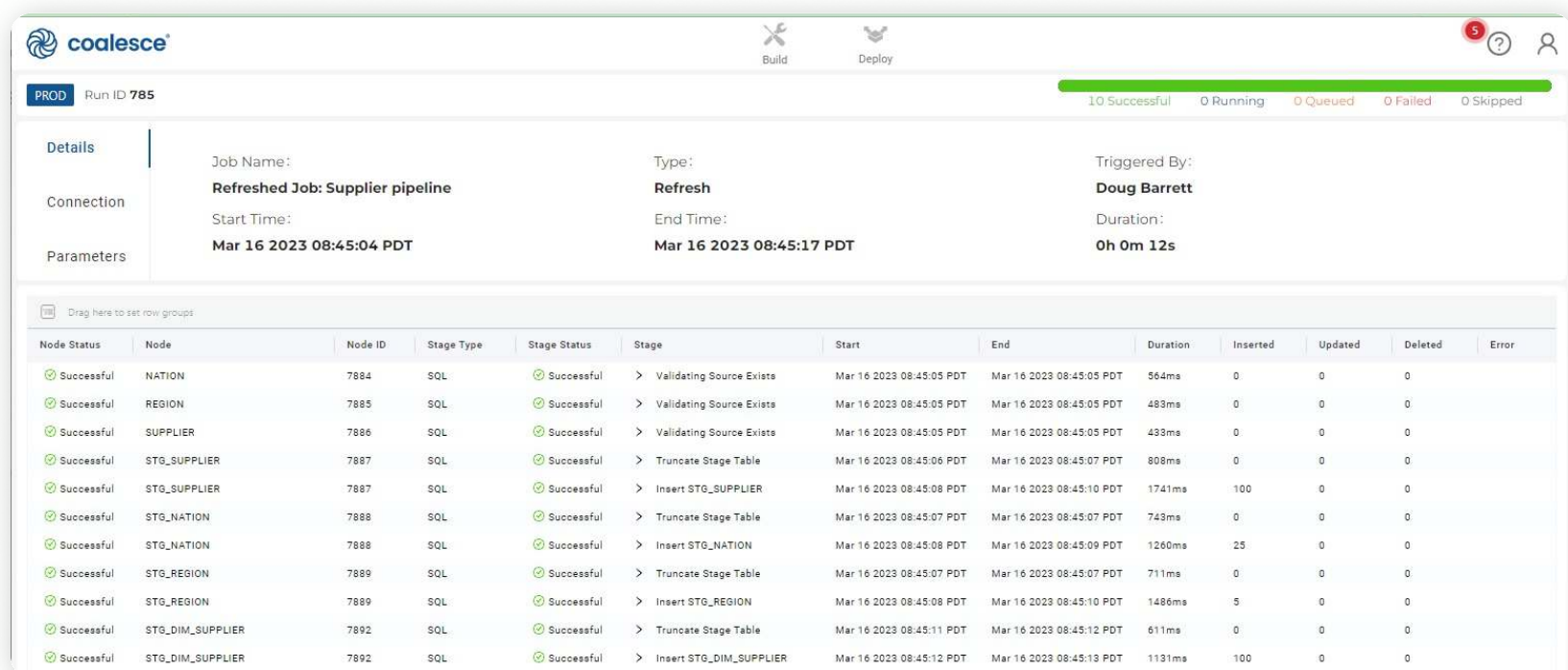
Jobs in Coalesce consist of tasks and dependencies which define a unit of work. Jobs are executed via our Web API (docs.coalesce.io/reference/startrun) or Command Line Utility (docs.coalesce.io/docs/cli-overview), allowing Coalesce to work with any Orchestration tool including Azure DevOps Pipelines, GitHub Actions or Airflow (docs.coalesce.io/docs/scheduling-overview).

Typically, Jobs are defined to process the data warehouse in layers and by source / subject depending on an event or the cadence required. For example:

- Persist Job 1 – Process data from Sales system through to Persistence Layer
- Persist Job n – Process data from n system through to Persistence Layer
- Conform Job 1 – Process conformed dimensions
- Mart Job 1 – Process Data Mart 1
- Mart Job n – Process Data Mart n

Dependencies between jobs can be defined in the Orchestration tool. Jobs can be run in parallel.

Job logs show which tasks were executed, the status of job and each task (success or failure), and any Database messages and metrics are recorded in Coalesce's Activity Log.



The screenshot displays the Coalesce interface for a job execution. The job name is 'Refreshed Job: Supplier pipeline', triggered by 'Doug Barrett' on 'Mar 16 2023 08:45:04 PDT'. The job duration is '0h 0m 12s'. The job status is '10 Successful', with 0 Running, 0 Queued, 0 Failed, and 0 Skipped. The table below shows the execution details for each stage.

Node Status	Node	Node ID	Stage Type	Stage Status	Stage	Start	End	Duration	Inserted	Updated	Deleted	Error
Successful	NATION	7884	SQL	Successful	> Validating Source Exists	Mar 16 2023 08:45:05 PDT	Mar 16 2023 08:45:05 PDT	564ms	0	0	0	
Successful	REGION	7885	SQL	Successful	> Validating Source Exists	Mar 16 2023 08:45:05 PDT	Mar 16 2023 08:45:05 PDT	483ms	0	0	0	
Successful	SUPPLIER	7886	SQL	Successful	> Validating Source Exists	Mar 16 2023 08:45:05 PDT	Mar 16 2023 08:45:05 PDT	433ms	0	0	0	
Successful	STG_SUPPLIER	7887	SQL	Successful	> Truncate Stage Table	Mar 16 2023 08:45:06 PDT	Mar 16 2023 08:45:07 PDT	808ms	0	0	0	
Successful	STG_SUPPLIER	7887	SQL	Successful	> Insert STG_SUPPLIER	Mar 16 2023 08:45:08 PDT	Mar 16 2023 08:45:10 PDT	1741ms	100	0	0	
Successful	STG_NATION	7888	SQL	Successful	> Truncate Stage Table	Mar 16 2023 08:45:07 PDT	Mar 16 2023 08:45:07 PDT	743ms	0	0	0	
Successful	STG_NATION	7888	SQL	Successful	> Insert STG_NATION	Mar 16 2023 08:45:08 PDT	Mar 16 2023 08:45:09 PDT	1260ms	25	0	0	
Successful	STG_REGION	7889	SQL	Successful	> Truncate Stage Table	Mar 16 2023 08:45:07 PDT	Mar 16 2023 08:45:07 PDT	711ms	0	0	0	
Successful	STG_REGION	7889	SQL	Successful	> Insert STG_REGION	Mar 16 2023 08:45:08 PDT	Mar 16 2023 08:45:10 PDT	1486ms	5	0	0	
Successful	STG_DIM_SUPPLIER	7892	SQL	Successful	> Truncate Stage Table	Mar 16 2023 08:45:11 PDT	Mar 16 2023 08:45:12 PDT	611ms	0	0	0	
Successful	STG_DIM_SUPPLIER	7892	SQL	Successful	> Insert STG_DIM_SUPPLIER	Mar 16 2023 08:45:12 PDT	Mar 16 2023 08:45:13 PDT	1131ms	100	0	0	

Job Execution Logs in Coalesce's Activity Feed

Conclusion

The steps for planning, building, and operating data pipelines we have outlined in this document will help you get started with your Snowflake data warehousing projects. Coalesce enables and automates built-in best practices that greatly accelerate implementation, with support for scale, while leveraging all features and capabilities of the Data Cloud.

Try Coalesce for free: coalesce.io/start-free/

Request a demo: coalesce.io/request-demo/

About the author



Doug Barrett

Principal Solutions Architect, Coalesce

As Principal Solutions Architect at Coalesce, Doug Barrett has over 25 years of experience working in data warehousing. Over the years Doug has run and delivered projects, designed models, and built processing frameworks with leading banks, retailers, airlines and companies around the world. Prior to Coalesce, Doug held consulting and solution architect roles at Clean Data, Microsoft and Wherescape.



About Coalesce

Founded in 2020, Coalesce is the only data transformation platform built for scale. Coalesce combines the speed of an intuitive graphical user interface (GUI), the flexibility of code, and the efficiency of automation, empowering its customers with increased data team productivity and insights. Based in San Francisco, Calif., Coalesce is backed by Emergence Capital, 11.2 Capital, GreatPoint Ventures, and Industry Ventures and supports customers worldwide. Learn more at coalesce.io.