O'REILLY®
Technical Guide

Accelerating Data Pipeline Development

Deliver Data Projects Faster Without Creating Tech Debt

Early Release
RAW & UNEDITED

Compliments of

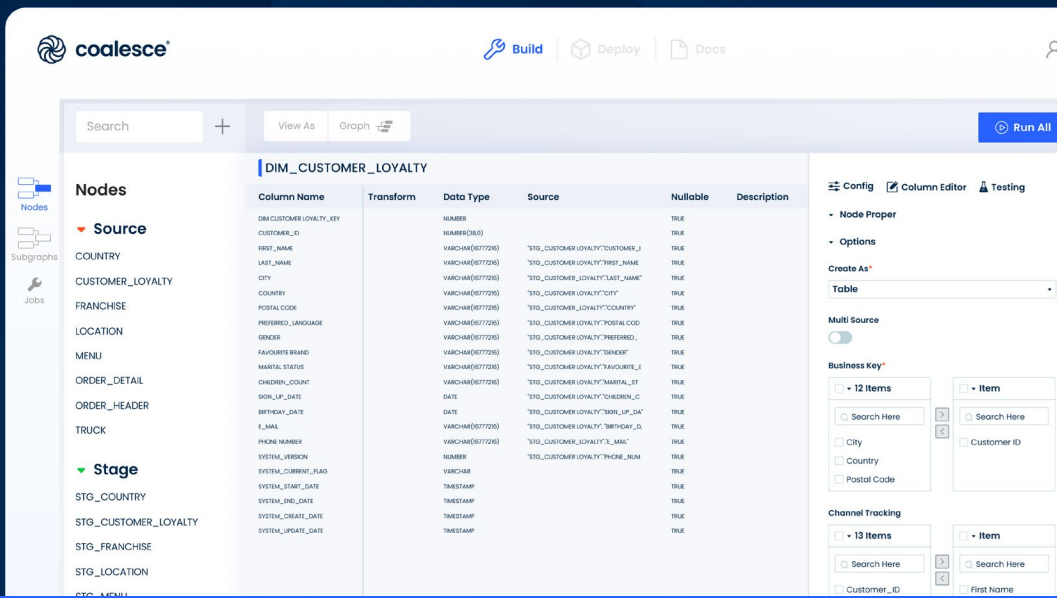
coalesce®

Josh Hall

coalesce®

Deliver data projects 10x faster – without creating tech debt

Build a supercharged data foundation and cut through the noise of data development with Coalesce.



Discover the future of data transformations at Coalesce.io

Accelerating Data Pipeline Development

Deliver Data Projects Faster Without Creating Tech Debt

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Josh Hall

O'REILLY®

Accelerating Data Pipeline Development
by Josh Hall

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Aaron Black
Development Editor: Melissa Potter
Production Editor: Kristen Brown
Interior Designer: David Futato
Cover Designer: Ellie Volckhausen
Illustrator: Kate Dullea

September 2025: First Edition

Revision History for the Early Release

2025-03-11: First Release
2025-04-02: Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9798341608740* for release details.

979-8-341-60874-0

[LSI]

Table of Contents

Brief Table of Contents (*Not Yet Final*)

CHAPTER 1

Getting Started in Coalesce

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

Just like the foundation that supports a house, Coalesce has foundational components that support your development of data products. This chapter will lay the groundwork for the rest of this guide, ensuring you have the tools you need to build end-to-end data pipelines. In it, you will learn about projects and workspaces, which are the core of the development experience in Coalesce. You will discover how to connect Coalesce to your data platform and make your data ready for development. You'll also see how you can work with the rest of your data team to build nodes for all your data transformation needs.

Equipped with this knowledge, you'll be able to get started developing your data in Coalesce and have a foundational understanding

for the rest of topics discussed in this guide. Let's start by learning about the user interface.

The User Interface

Coalesce provides a graphical user interface (GUI) that enables you to develop your data while giving you the flexibility to write SQL. The interface is divided into different segments which provide support for different functions throughout the data development lifecycle. These segments include:

- The Projects Page
- The Build Interface
- The Deploy Interface
- The Docs Interface

You also have the ability to manage your Coalesce organization and users from the user menu interface using Org and User Settings. In this section, you'll explore each of the segments of the Coalesce interface.

The Projects Page

The projects page is the default landing page when logging into Coalesce. This page will display any of your organization's projects that you have access to. Projects in Coalesce give you the flexibility to organize your data initiatives for a specific purpose of team goal, as shown in Figure 1-1. Don't worry if you can't make out all the details here, we'll dig into these screens in more detail shortly.

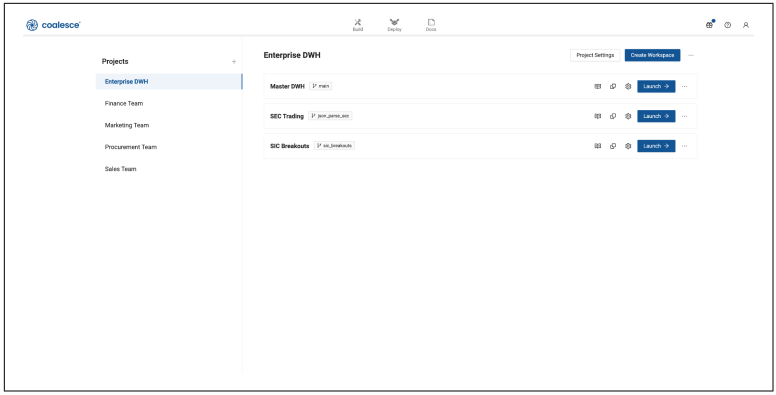


Figure 1-1. The projects page of Coalesce.

The Build Interface

The build interface is where you will spend time developing your data products and building node graphs and pipelines as shown in Figure 1-2. It can be accessed by launching any workspace from the projects page. Users can easily manage each aspect of a data pipeline from the build interface, including creating jobs and subgraphs.

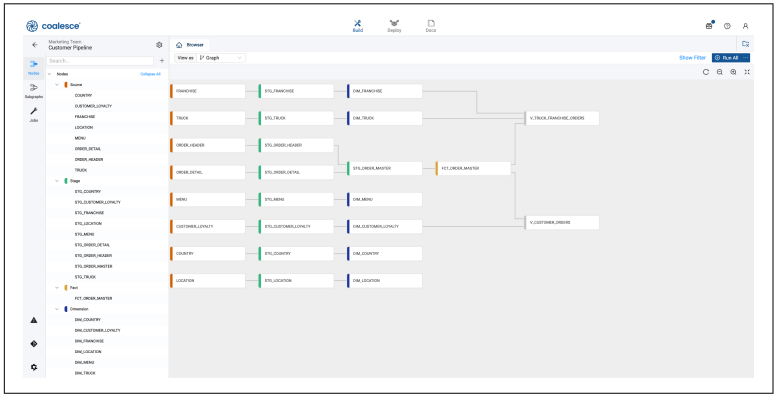


Figure 1-2. The build interface in Coalesce displaying various nodes. Don't worry if you can't make out all the details here, we'll dig into these screens in more detail shortly

The Deploy Interface

The deploy interface is where you can deploy your data projects from the desired state in your git repository, and see a history of your pipeline's deployments and refreshes from the feed on the right

side of the screen as shown in Figure 1-3. This interface contains a dashboard for any environment you create, allowing you to see the current status of job runs as well as allowing you to schedule job refreshes.

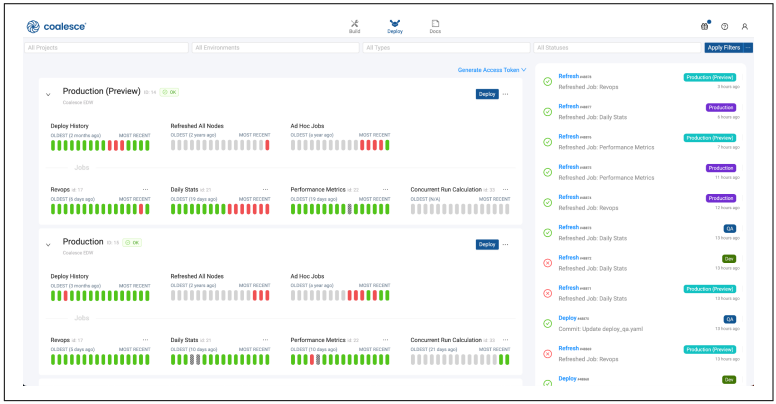


Figure 1-3. The deploy interface where you can manage and monitor your data projects. Don't worry if you can't make out all the details here, we'll dig into these screens in more detail shortly

The Docs Interface

The docs interface captures automatic, real-time documentation about each project and environment in your Coalesce organization. You can find information about each project created, such as the database and schema, column names and descriptions, and even data definition language (DDL) and data manipulation language (DML) as shown in Figure 1-4.

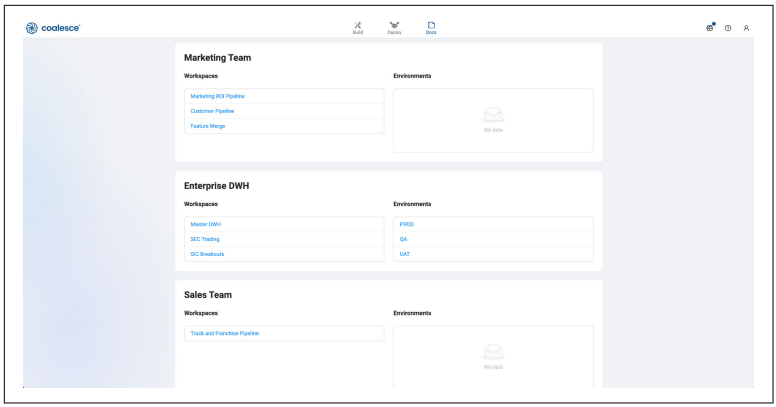


Figure 1-4. The docs interface displaying all of the workspaces and environments that have been documented

The User Menu

As a user within Coalesce, you will have access to the user menu. The user menu can be accessed in the upper right hand corner, denoted by the user icon as shown in Figure 1-5. Within the user menu are the Organization and User settings of your organization.

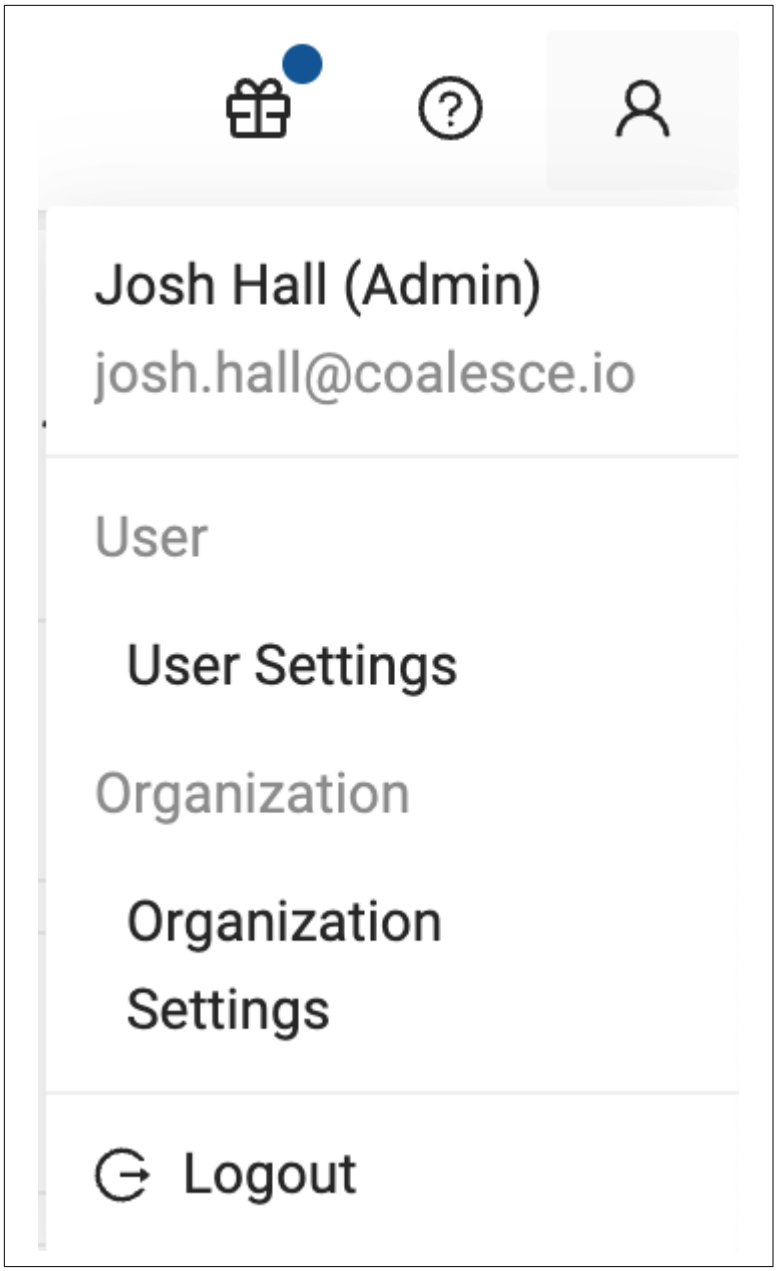


Figure 1-5. The user menu opened to display User and Organization settings

Organization settings provide management access to the following controls for your Coalesce organization:

- User management – adding, removing, or modifying users
- Single Sign-On
- Preferences – such as the Coalesce parser sample size

User settings provide you with the ability to manage your individual user with the following controls:

- Configuration of your Git settings
- Support information about your Coalesce account
- Changing your password

With the ability to navigate the Coalesce interface, you can now dive into setting up your first Coalesce project for developing your data.

Projects

You learned about the projects page in the previous section, but now it's time to take a step further to learn more about projects. Projects give you the ability to organize your data development in a structured way, similar to how folders allow you to organize documents in Google Drive. In this section, you will learn how to use projects, as well as how to set up a project for your own data development.

The Purpose of Projects

Projects provide multiple advantages for data teams developing their data. The first of these advantages is architectural. By utilizing projects, you can decide how your data development processes should be architected.

For some data teams, this means organizing projects by the initiatives that the data team is working on. Others may want to implement a data mesh pattern and use projects to separate each domain of the business. Still others may want different teams within the organization to be managed through separate projects as shown in Figure 1-6. Regardless of the organization pattern, you must have at least one project in order to develop your data.

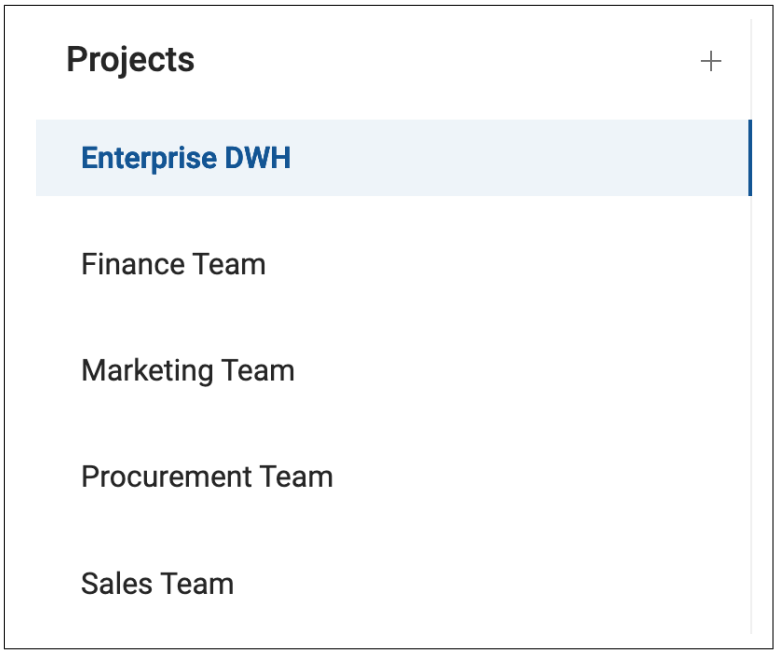


Figure 1-6. Projects in Coalesce managed by the team working on the project

Another advantage of projects is version control. Each project in Coalesce is integrated with your version control system such as GitHub. You will integrate a git repository for each project that you create. This allows each project to be managed separately from the others while providing version control for the specific purpose of the project.

While it is possible to skip this integration, your development experience will be limited to a singular workspace. Coalesce does not recommend developing this way.

The last advantage we'll discuss is role based access control, or RBAC. RBAC allows your Coalesce administrators to determine which users should have access to which projects, as well as determining which level of access each user should have. This provides granular control for all of your data development initiatives.

How to Set Up a Project

Now that you know when to use a project, let's talk about how to set one up. If you have never configured a Git account in Coalesce before, you will need to do this first. To configure a Git account in Coalesce, navigate to the user settings. Within the version control section, select Add New Account. You can follow the instructions on the setup modal to provide the information necessary to configure your Git account as shown in Figure 1-7.

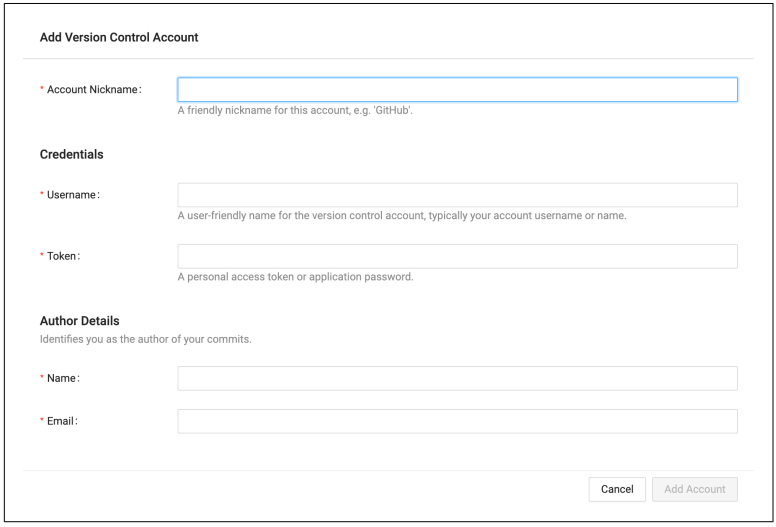


Figure 1-7. The Git setup modal in Coalesce

From the projects page, click on the plus + button next to the Projects header. This will open the project configuration workflow. Here you'll give your project a meaningful name, select the data platform you want to connect Coalesce to, and provide any descriptive information as shown in Figure 1-8.

Figure 1-8. The Project set up workflow where you select your data platform and supply a name and description

Next, you'll need to provide the Git repository URL from your version control system. You can skip this step to create a project without version control, but I don't recommend this. Once you supply the Git repository URL, you can select a Git account configuration from Coalesce. Once you have selected your Git account, your setup is complete and you can complete the project setup workflow.

Workspaces

With your new knowledge of projects, you can now move on to setting up a workspace for your data development projects. A workspace is a sandbox environment where you can complete the development of your data. Each workspace has its own graph, storage locations, macros, node types, data platform connection configuration, and Git branch – all of which you will learn about in subsequent sections. You can create multiple workspaces to work on different tasks and merge them into your codebase. In order to begin building your data pipelines, you will need to connect your workspace to your data platform. Let's go over how to do that now.

Within any project, select the Create Workspace button in the upper right corner of the project – this will open the workspace creation workflow. You will be asked for a workspace name and description. Next, you will select the branch and commit you want to create your new workspace from. For example, you may want to create a workspace from your main branch in order to design a new forecasting

pipeline in your data warehouse. Once you have selected the branch and commit, you will supply the name of the new branch you wish to create and finish the creation of the workspace.

With a workspace created, you can now connect it to your data platform. By clicking on the gear cog icon next to the workspace Launch button, you can provide the information about your data platform connection. In the case of the Figure 1-9, this workspace is connected to Snowflake, but Coalesce supports multiple different data platforms.

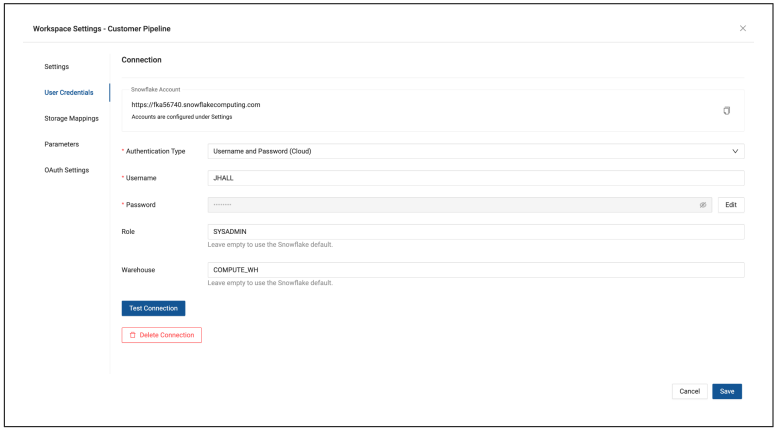


Figure 1-9. Connecting your workspace to your data platform. In this case, Snowflake

Coalesce supports multiple different authentication types. The two most common are OAuth and Username and Password. Once you have supplied Coalesce with your account information and connection criteria, you can test the connection to your data platform. Once successful, you are ready to map your workspace to the data that exists in your data platform.

Once you have connected your workspace to your data platform, you can select the blue Launch button to enter the build interface. Before you can begin your data development in the build interface, there are a few more configuration items to complete. You can find these items in the Build and Workspace settings. We'll quickly explore each of these.

Build Settings

The build settings page can be accessed by clicking the gear cog icon in the lower left corner of the build interface. Within the build settings you can manage all aspects related to development within your workspace. This includes:

- Storage Locations and Storage Mappings
- Development Workspaces
- Environments
- Macros
- Node Types
- Packages

Each of these items has their own settings which can be configured by selecting each item.

Workspace Settings

Your workspace settings provide management for the connection to your data platform. These workspace settings can be accessed in one of two ways. You can either select the gear icon next to the workspace name, or select the same icon next to the workspace in the Workspace selection from the Build Settings as shown in Figure 1-10.

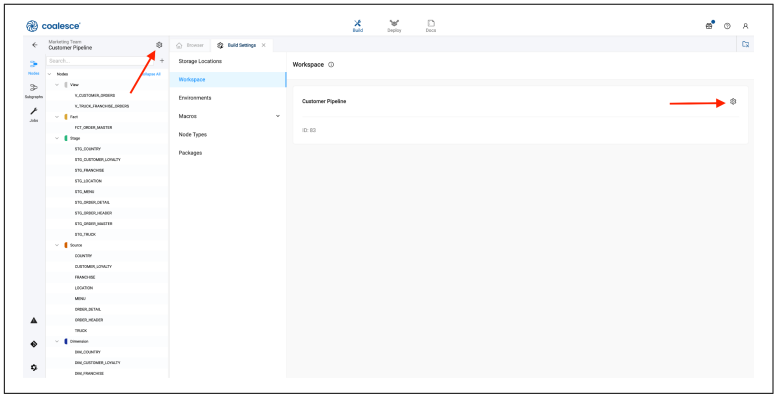


Figure 1-10. Where to open the workspace settings inside of a workspace

Now that you know where all of the settings are available in your workspace, you can finish the final configuration of your workspace to begin building your data pipelines.

Storage

So far you have created a project and workspace and connected the workspace to your data platform. But now you need to tell your workspace what data in your platform you want to develop with. This is where storage locations and storage mappings come into play.

Storage Locations

Storage locations are a logical representation of a database and schema in your data platform. You can think of them as the glossary that points to the chapters in a book. Storage locations themselves don't contain any data or perform any action on their own. Instead, they act as logical containers for the databases and schemas you want to use in your data pipeline development.

For example, you may have a database and two schemas within that database which contain source data for a pipeline you wish to develop. You want the results from your data pipeline to be output in a different database and schema. In this case, you could use three storage locations, two for the source data and one for the output or target destination as shown in Figure 1-11.

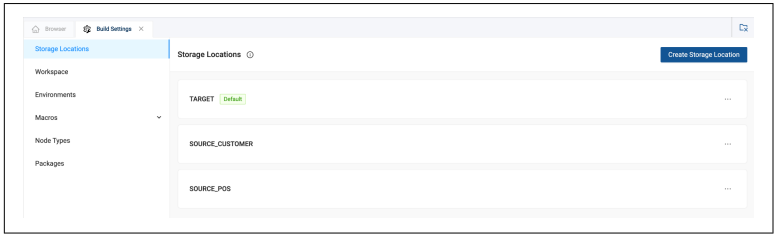


Figure 1-11. Storage locations created to be mapped to physical destinations in your data platform

Storage locations can also be used for other use cases as well. For example, if your organization uses a medallion architecture, you can have a storage location for each level of your architecture i.e. bronze, silver, gold. Or if you leverage a staging layer in your data pipelines,

you could have a staging storage location where all staging tables are created in the same location.

You will notice that there is always a *default* storage location. This is the location that, unless configured otherwise, all tables in your data pipeline will be created within by default.

It's important to note that *storage locations cannot be renamed once they are created*.

Once your storage locations are created, you can map each storage location to a physical destination in your data platform using storage mappings.

Storage Mappings

As you just learned, storage locations are just logical containers that can point to physical locations in your data platform. Storage mappings are what tie your storage locations to a database and schema in your data platform. To configure storage mappings for your workspace, you will need to access your workspace settings and select Storage Mappings.

Each of the storage locations you have created will show up as an item to be mapped to your data platform. For each storage location, you can select the database and schema that you want each to point to as shown in Figure 1-12. Ensure that your default storage location is mapped to a database and schema where you expect your tables to be output.

Figure 1-12. Mapping physical databases and schemas to the storage locations previously created

Storage mappings can be updated at any time from the workspace settings. You can also add more storage locations and map them in the same way described here. For this guide, we will be using a fictional foodtruck company dataset for our illustrations,where our data sources are the customer and POS data and our target storage location is a development database and schema. We can deploy this to a production environment later on.

Adding Users

You now have a fully configured and ready for development project and workspace. With your data ready for development, you can add your team into your Coalesce account to collaborate alongside them. To add users, you will need to access the Org settings from the User Menu. As a Coalesce administrator, you will be able to add users to your Coalesce account and assign them appropriate roles.

Table 1-1 provides guide to user roles in Coalesce and the permissions that they contain.

Table 1-1. Coalesce User Roles and Associated Permissions

Role	Permissions Summary	Recommended For
Organization Administrator	The creator of the Coalesce App is automatically assigned as organization administrator. Only organization administrators can add other users, including other organization administrators. They have full access to all functionality in Coalesce.	Full administrative control
Organization Contributor	They can't add new users to the organization. They have access to read documentation, create API tokens, user settings, and Git account information. They will be able to set up a project, configure Git, add members to projects,and oversee work. They'll only have access to the projects they create themselves. If there are multiple organization contributors, they will need to share access with the organization contributor.	Managers who decide how each person will contribute to a project.
Organization Member	This is the default role. They can edit Git account information, create API tokens, and read documentation.	Default Role

Now that your users are added into your workspace with the proper permissions, your team can begin building data products by adding data sources

Adding Data Sources

With your team ready to collaborate in Coalesce and your workspace ready to develop your data, you can begin adding data sources to your graph. You can do this by launching your workspace and selecting the plus + button in the upper left corner of the build interface. This will open the add data sources modal, which will display all of the storage locations you mapped your data to, and the objects available in those locations as shown in Figure 1-13.

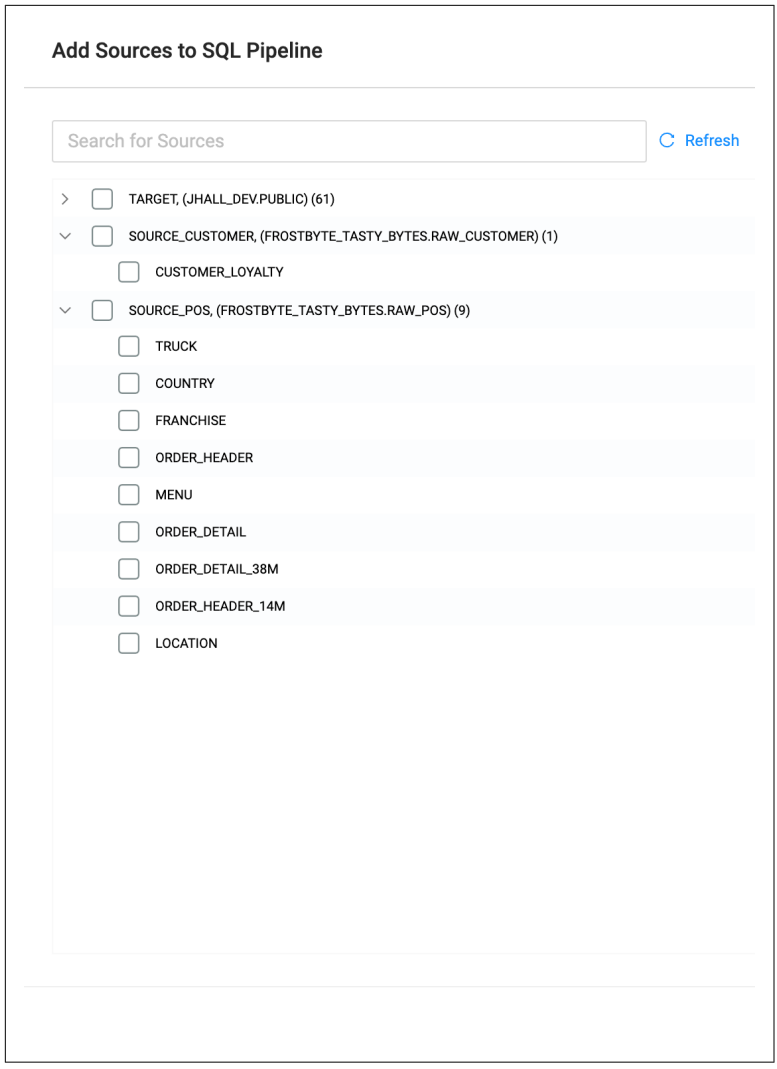


Figure 1-13. Data sources modal showing all of the objects available from each of the storage locations configured.

You can select as many or few data objects as you want to add to your data pipeline. Once you have selected the data sources you want to work with, you can add them to our graph. Coalesce will add each object into your graph as a node. In the next chapter of this guide, we will discuss what a node is in detail, but for now, all you

need to know is that it's a visual representation of the objects you have added into your graph.

Building on Your Foundation

This chapter laid the foundation for all the components you will be using in Coalesce. In it you learned how to navigate the user interface and how each segment of the interface is used. You learned about the purpose of projects and how to set one up. You also learned how to create a workspace and connect to your data platform. At the end of the chapter, you saw how to bring data sources into your graph. At this point, you are ready to begin developing your data.

However, before you begin building a data pipeline, it's important to understand the core concepts of how to develop your data in Coalesce. In the next chapter, we'll explore everything you need to know to begin your journey building data pipelines and applying all of the knowledge from this chapter, to the concepts of pipeline development in Coalesce.

CHAPTER 2
Coalesce Core Concepts

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

In the previous chapter we laid the foundation for your understanding of Coalesce components, such as projects, workspaces, and storage locations. In this chapter, we will lay the building blocks on top of your foundation, by providing you with the core concepts that empower Coalesce data development. You will learn why column-aware architecture is important within data transformations, what nodes are and how they harness column-aware architecture, the data pipeline approach, and managing data development.

At the end of the chapter, you'll walk away with the knowledge that will provide you with the framework you need to understand and develop your data on the platform. And there is no better place to start than understanding column-aware architecture.

Column-aware Architecture

Data processing workloads in modern data platforms don't just operate on the scale of thousands of database tables; they run on hundreds of thousands—even millions—of columns. Coalesce is built from the ground up to automate data transformations while keeping a code-first approach and flexible interface. This all starts with a column-aware architecture.

But what is column-aware architecture? Column-aware architecture, or being column-aware, is an approach to managing data transformation with an understanding of columns and how they are connected. With this understanding, Coalesce provides automated column-level lineage, while enabling the creation and maintenance of database objects at scale. This column-aware architecture captures metadata for each object created, allowing you to leverage incredible development speed and agility in your data transformation workloads.

By using column-aware architecture, Coalesce shifts the paradigm of traditional data transformation processes to an automated, reusable, and scalable approach. We'll see how this translates to the foundational building blocks of Coalesce (nodes), but first, let's dig deeper into the benefits of building a data transformation platform using column-aware architecture.

Data Patterns

Column-aware architecture is the key to standardizing how transformations are applied, how tables are structured, and how columns are logically connected. This standardization can be referred to as a pattern, which is a reusable step in a data transformation process that represents a logical transformation. This can include:

- Incremental & processing logic
- Materialization logic
- Deployment logic

These data patterns are essential for the creation, management, and accessibility of data, especially at enterprise scale. Coalesce provides a platform to rapidly implement these data patterns by leveraging metadata at the column level. With column-aware metadata, you

can build a single reusable data pattern that can be applied across any of your data transformations.

Take, for example, a simple type 2 slowly changing dimension–an industry standard for tracking historical data, such as a customer's current address and what it was six years ago, and every change in between. Writing the complex SQL to deliver this functionality could take hundreds of lines of code. Because Coalesce is column-aware, this can be defined once and then reused as many times as required without the need for writing the code each time.

In Coalesce, column-aware data patterns streamline complex, time-consuming manual coding tasks. This speeds up data processing and lets you focus on broader strategy while applying reusable logic with confidence..

Impact Analysis and Lineage

Having data patterns is powerful, but if those patterns aren't coupled with the ability to understand and manage your entire pipeline in one place, you may end up spending all of the time you saved building patterns having to understand how changes impact your pipeline. This is a second powerful benefit of using column-aware architecture: this column-awareness enables complete impact analysis and lineage at the column level.

Imagine you've just deployed a pipeline that powers critical stakeholder dashboards. It's Monday morning, and you wake up to an email from your SaaS ETL provider: they've changed the schemas of several tables in your pipeline. Panic sets in. Which columns in my pipeline are impacted? Are dashboards already broken? How many transformations are affected?

With column-awareness, you can quickly see how changes to your data affect everything downstream as seen in Figure 2-1. You'll know exactly what's been impacted and can fix issues before they become problems. You can see exactly how each object and column affects your pipeline at any moment, no guesswork or managing dozens of SQL tabs.

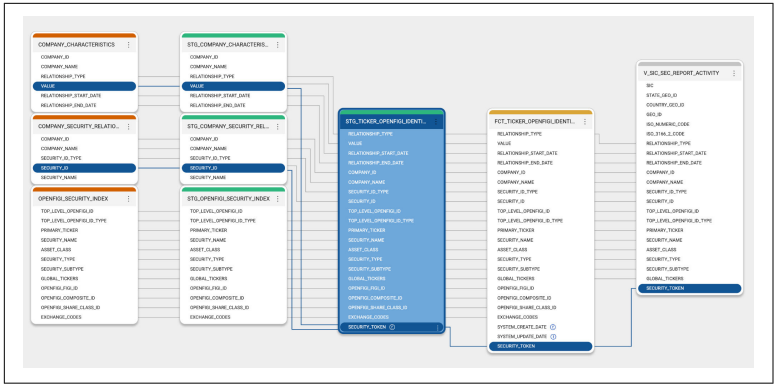


Figure 2-1. Column level lineage showing the impact of every column throughout a pipeline.

Additionally, with column-awareness, you can perform bulk operations directly at column granularity and across your data platform, making source data changes or business logic changes easy and straightforward to implement. For a data consumer, Coalesce provides clear documentation at the column level, showing where the data came from and how it was calculated. This transparency helps build trust across the organization.

Scale and Governance

Column-awareness drives efficiency and accuracy in data management, even at scale—across hundreds of thousands of columns, multiple teams, environments, and the entire business.

Column-aware state management minimizes the risk of data loss and errors by enabling in-place, column-level modifications instead of requiring full table re-creations. It also offers detailed column-level visibility into changes over time, a critical aspect of effective DataOps and governance practices.

For instance, in a deployment with thousands of columns, Coalesce eliminates the need to build complex architectures to handle changes. Instead, you can manage in-place edits out of the box This approach reduces costs, enhances deployment visibility and planning, and fosters trust among data consumers throughout the organization.

This column awareness enables column propagation, allowing you to add or remove columns across the entire pipeline with ease

through the user interface. This functionality streamlines column management across your project, making column awareness a core advantage.

Now that you have an understanding of column-aware architecture in Coalesce, let's dive into the foundation of data development: nodes.

Nodes

Nodes are the core components used to build data pipelines in Coalesce. In Chapter 1 we defined a node as a visual representation of an object (table, view). Now that you have an understanding of pattern based development, we can provide a more precise definition.

A node is a visual representation of an object within your data platform as seen in Figure 2-2. It serves as a building block for constructing data pipelines and leverages data patterns to automate your data transformations. Nodes are classified as node types, such as a stage or fact node type–more on this shortly!

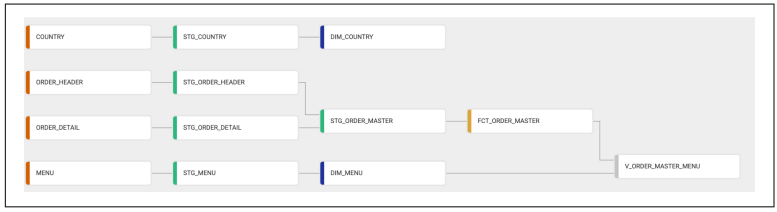


Figure 2-2. Nodes in the build interface representing a data pipeline.

Nodes enable pattern-based development by allowing you to define a transformation once and repeatedly apply it, streamlining automation and ensuring consistency across your pipeline. You can begin to see how leveraging nodes can accelerate development while ensuring quality and consistency across your pipelines, as everyone works from a shared pattern or standard. To build effectively with nodes, it's essential to understand how node types are created and function. Let's take a closer look at what makes up a node.

Node Architecture

Node types consist of three components: a node definition, a create template, and a run template. Each of these components performs a

specific role in the representation and execution of a node type. Let's start with the node definition.

Node definition

The node definition defines the attributes available within a node type, such as naming conventions and node type colors. It also specifies the UI elements used to configure each individual instance of the node type. For example, if you want to build a stage node type for creating a consistent staging layer in your pipeline, the node definition would define the naming convention for the node each time it's used. However, each instance of the stage node can be configured differently, based on the options provided through the UI elements in the node's configuration. Such as one stage node type being materialized as a view and another being materialized as a table.

The node definition is defined using YAML as seen in Figure 2-3.

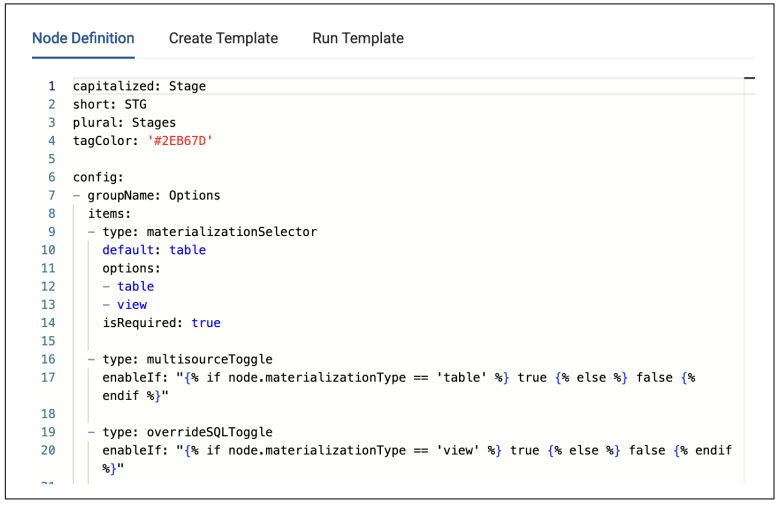


Figure 2-3. YAML used in the node definition to define the attributes of the node type.

The Coalesce documentation contains information on the configuration options available to include in your node definition YAML file as seen in Figure 2-4.

Figure 2-4. Coalesce documentation representing the various configuration options for the node definition of a node type.

Once you have your node definition defined, you can configure the Create and Run templates.

Create Template

The create template for a node type defines the logic used to automate the creation of the object. Typically, this involves Data Definition Language (DDL) that leverages attributes from the node definition—such as the selected materialization type—to execute a CREATE statement in SQL.

You can define create templates using a combination of SQL and Jinja, as shown in Figure 2-5.

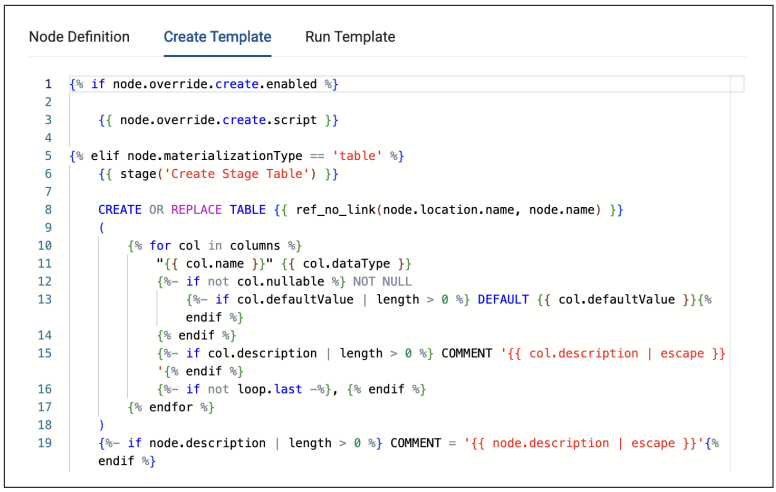


Figure 2-5. The create template representing the SQL and Jinja used to automate your DDL.

After you define the logic, Coalesce runs the DDL for each object instance in your pipeline. Executing the create template creates the object in your cloud provider automatically.. It is important to note that the create template is *only* creating the object, it is not loading data into the object. In order to insert data into the object, you need the run template!

Run Template

The final component of a node type is the run template. This is where you define the logic to automate the execution of your transformations. Typically, it involves Data Manipulation Language (DML) that leverages Coalesce's column-aware architecture to execute SQL and populate objects with data.

Like create templates, run templates are defined using a combination of SQL and Jinja as shown in Figure 2-6.

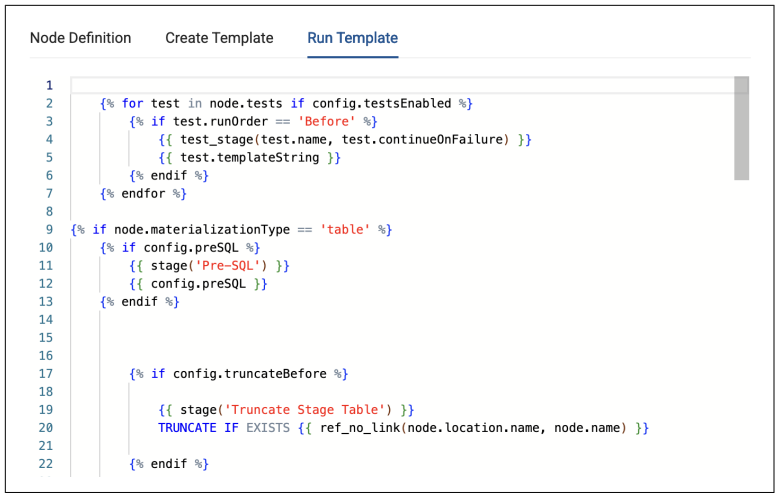


Figure 2-6. The run template representing the SQL and Jinja used to automate your DML.

With the logic defined, Coalesce will automatically run the corresponding DML for each node type. This will populate the objects in your data platform with data based on the logic and upstream data coming from your data pipeline. It's important to note that some node types may not need a run template, such as a view node type, which is just storing a query to be run in the future. A view is never actually populated with data.

Now that you understand the components of a node type and how you can create reusable patterns, let's explore the advantages of developing with nodes.

Importance of nodes

Up to this point, we've seen how column-aware architecture drives pattern-based development, accelerating data transformations within Coalesce. Now, it's important to bring everything together and understand how these concepts translate into more efficient pipeline development.

Standardization

We've already discussed how node types establish a framework for consistent data development across your team. Now, let's delve into

why this standardization is essential for long-term, scalable pipeline development.

Standardization ensures that everyone operates from the same foundation, automatically and consistently. There's no need for new custom code, special permissions, or isolated environments. A node type is defined once, and it's ready to use repeatedly without additional setup.

This standardization enables your team to embed best practices directly into the nodes. Each time a node runs, you can trust it to perform exactly as intended. It also allows you to optimize your code, ensuring that each instance of a node type runs efficiently and minimizes compute resource usage within your data platform.

Standardization streamlines development by reducing repetitive object creation and manual coding. This allows your team to focus on data transformation and modeling rather than boilerplate setup. By minimizing development overhead, you can build and deploy pipelines more efficiently.

As discussed earlier, you can define a dimension node type to support type 2 slowly changing dimensions (SCDs). Whenever you need to implement a type 2 SCD, you can add the dimension node to your pipeline and configure it with just a few clicks as seen in Figure 2-7. In seconds, you can automate the execution of a type 2 SCD, eliminating the need to write code from scratch or manually adapt copied code to your specific tables.

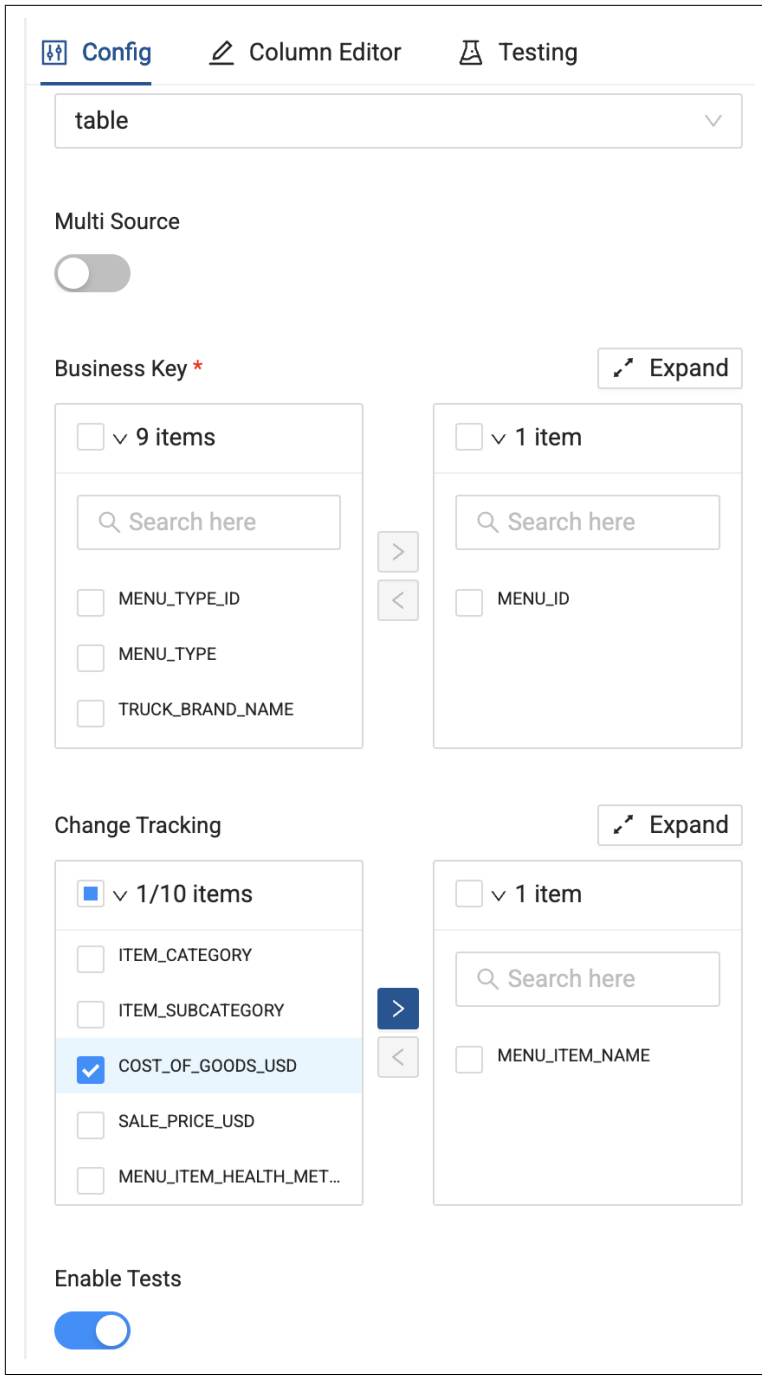


Figure 2-7. The configuration options for a dimension node, allowing you to set up type 2 SCD capabilities in seconds.

By using standardization in your pipeline development, you gain all of the advantages discussed here. This does not mean that you are giving up flexibility, as you'll see in the next section.

Flexibility

Coalesce provides multiple built-in node types, including a dimension node with preconfigured Type 2 SCD options, to streamline development. While these predefined options simplify automation, data projects often require customization.

Coalesce supports both custom code and a GUI-driven interface (Figure 2-8), allowing you to define node behavior precisely while maintaining an efficient and user-friendly development experience.

Figure 2-8. Coalesce provides you the ability to define your own data patterns and write custom SQL all through a power user interface.

Coalesce enables you to create User Defined Nodes (UDNs), allowing you to configure nodes tailored to the specific needs of your data development. Additionally, Coalesce provides a variety of prebuilt node types available on the Coalesce Marketplace, designed to address a wide range of use cases. We'll learn more about Coalesce Marketplace in the next chapter.

Independent blocks of logic

You've likely encountered SQL transformations that span hundreds of lines, filled with multiple CTEs and repetitive code that violates the DRY (Don't Repeat Yourself) principle. Coalesce node types are designed to address this by breaking transformations into logical, reusable blocks.

Instead of bundling multiple CTEs and subqueries into one sprawling query, Coalesce allows you to handle each logical task in its own

node. This means each CTE in a large query can become a separate node in your pipeline.

You might be thinking that building pipelines this way will result in more objects compared to consolidating everything into a single query—and you're right! But typically, this is only a problem because more objects typically means more development time and management. But, because Coalesce relies on standardized, reusable nodes, the building process is incredibly fast. And with column-aware architecture in place, managing a pipeline at any scale becomes simple. As we'll explore in the next section, there are significant advantages to designing pipelines using this pipeline approach rather than relying on complex CTEs.

The Pipeline Development Approach

Complex data processing often involves breaking tasks into sequential steps, where the output of one step feeds into the input of the next. While Common Table Expressions (CTEs) can achieve this, Coalesce takes a modular, pipeline-based approach that offers significant advantages over traditional CTE development.

Transparency
: One of the primary benefits of a pipeline approach is improved transparency. Instead of consolidating all logic into a single query, Coalesce uses nodes to represent individual logical blocks. This modularity makes it easy to navigate your pipeline and understand the function of each step. Additionally, you can run individual nodes independently to view their results in context, simplifying analysis and debugging.

Troubleshooting
: The transparency of pipeline logic also simplifies troubleshooting. Debugging a monolithic SQL query with hundreds of lines and multiple CTEs can be tedious and error-prone. In Coalesce, each logical step is a standalone node, allowing for a more straightforward process to identify and resolve errors or data issues. This modular design reduces the overhead required to troubleshoot and maintain data pipelines.

Testing
: Coalesce facilitates out of the box and SQL-based testing in each node in the pipeline as shown in Figure 2-9. For example,

you can validate uniqueness, check for null values, or run other data quality checks at any step. By catching unsupported data or logical errors early in the pipeline, you can prevent issues from propagating to downstream processes, saving time and effort.

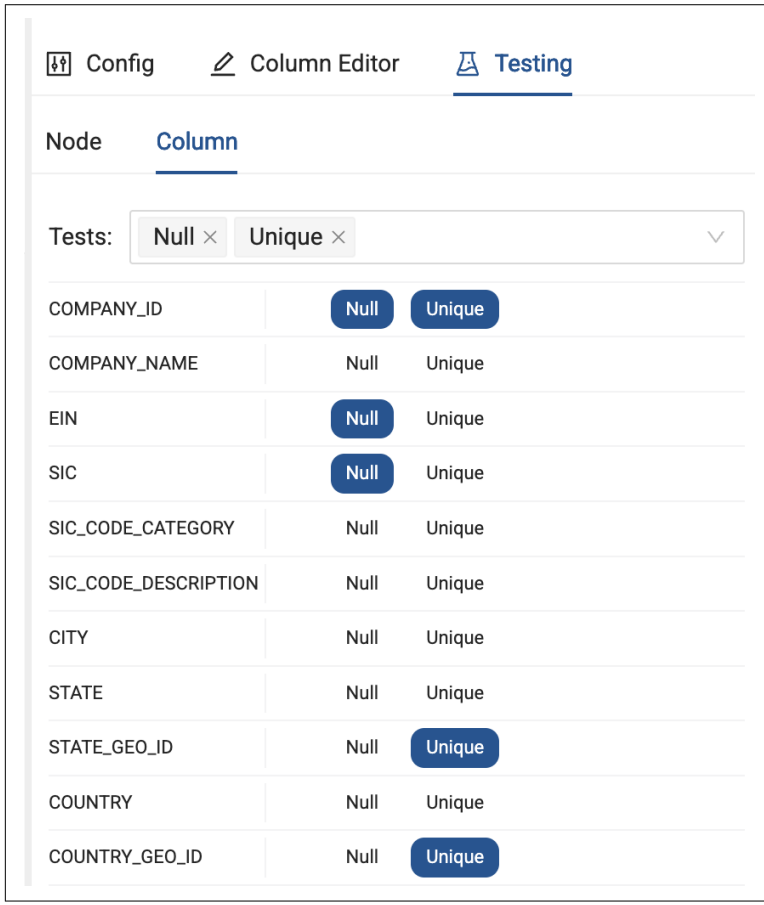


Figure 2-9. Out of the box tests that can be applied easily to any column in the node.

Developing pipelines in this way allows complete control over each element, while providing a streamlined approach to development.

Reusability

Breaking a data pipeline into nodes promotes reusability of logic across the pipeline. For instance, if you create a node to deduplicate

records in an orders table, downstream nodes can reference and reuse this logic without reprocessing. This eliminates redundant computation and ensures consistency across the pipeline.

In contrast, using CTEs often requires duplicating logic across multiple queries, which is harder to manage and less efficient.

Developing with a pipeline approach offers substantial advantages in transparency, troubleshooting, testing, and reusability. While this method may result in more objects in your data platform, Coalesce's column-aware architecture and node standardization make managing pipelines of any size seamless and efficient.

The Development Workflow

So far, we've explored the fundamentals of nodes and the advantages of building pipelines with them in Coalesce. But what does the actual development workflow look like? While we'll dive deeper into pipeline construction in the next chapter, there are two essential concepts to understand for developing your data in Coalesce: *workspace development* and *environments*.

Workspace Development
: Each workspace in Coalesce comes with a build interface designed for data development. When working within a workspace, each user will have their own set of credentials. Typically, development occurs in a sandbox or personal space within your data platform. This means that any objects you create are isolated to your storage location, allowing you to experiment and iterate without impacting shared environments.

  Think of a workspace as your personal sandbox for developing and testing data pipelines. Once you've finalized your pipeline, you can deploy your work to an environment for broader use.

Environments
: Coalesce provides the use of environments for deploying data pipelines, allowing for a structured workflow from development to production. After finalizing changes in the workspace, commit them to a Git branch and deploy to the target environment.

  Environments in Coalesce are tied to designated locations in your data platform, such as specific databases or schemas through storage locations and storage mappings. For best prac-

tices, Coalesce supports multiple environments, such as *QA* and *PROD*, allowing you to test and validate your work before deploying it to production. Figure 2-10 illustrates an example of a typical environment setup.

Figure 2-10. The deploy interface in Coalesce, where you can deploy your pipelines to higher environments in your data platform.

Knowledge Sync Complete

You are now equipped with the foundational knowledge needed to start developing your data in Coalesce. In this chapter, we covered column-aware architecture, how it supports pattern-based development through nodes, and why nodes serve as the optimal building blocks for building pipelines.

If you encounter challenges as you progress through this guide, feel free to revisit Chapters 1 and 2 for a refresher.

Up next, we'll walk you through the process of building a data pipeline—from data source to insight-ready tables. See you in the next chapter!

CHAPTER 3

Building Data Pipelines in Coalesce

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 3rd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

So far, you've spent time learning to set up Coalesce and explored core components and concepts of the platform, such as projects, workspaces, node types, storage locations and mappings, and column-aware architecture. In this chapter, you'll put this knowledge to work by learning how to build data pipelines in Coalesce.

In this chapter you'll learn how to add data sources, create nodes, and write SQL to transform data directly within any node. You'll also master functionality like creating joins, bulk-editing columns, and applying tests to ensure data quality. Plus, you'll discover how Coalesce Marketplace enhances your pipeline with powerful extensions.

Let's get started!

The Build Interface

In Chapter 1, you briefly learned about the Build Interface, which is where you will be spending your time in this chapter. The Build Interface is where you will develop your data products and build node graphs and pipelines. You can assess it by launching any workspace from the projects page as seen in Figure 3-1.

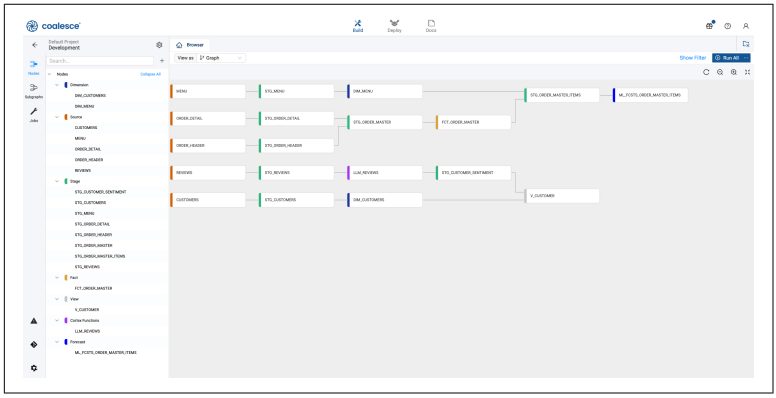


Figure 3-1. The Build Interface in Coalesce, displaying nodes organized in the form of a pipeline.

The Build Interface contains all of the necessary components to build data pipelines. In the sidebar in the upper left side of the interface, you'll see navigation options for Nodes, Subgraphs, and Jobs. You'll learn about subgraphs and jobs more in chapter 4, as the focus of this chapter is on building pipelines with nodes.

In the lower left corner of the sidebar, you'll have navigation for your Problem Scanner, which will alert you on any issues or notifications you should be aware of within your data pipeline. You'll also be able to access your Git Integration to easily version control your work. Finally, you'll see the Build Settings cog, which contains all of the settings for the workspace you are building in, as seen in Figure 3-2, such as the storage locations you learned about in Chapter 1. Throughout the rest of this guide, we'll explore each of the items located in the Build Settings so don't worry if you don't know what each of these items mean.

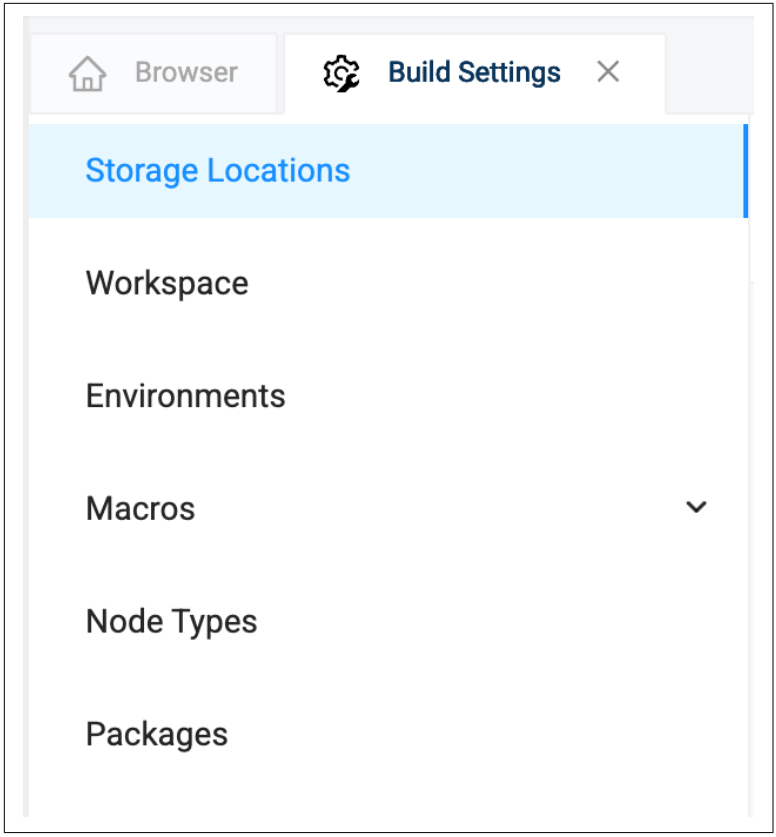


Figure 3-2. The Build Settings available in each Workspace in Coalesce.

The Build Interface also contains the Workspace Settings for the workspace you are working in. You learned all about the Workspace Settings in Chapter 1, when we discussed connecting to your data platform, but you can easily access those settings from the Workspace Settings Cog in the upper left corner of the screen, as seen in Figure 3-3. You can also find them in the Workspace line item in the Build Settings.

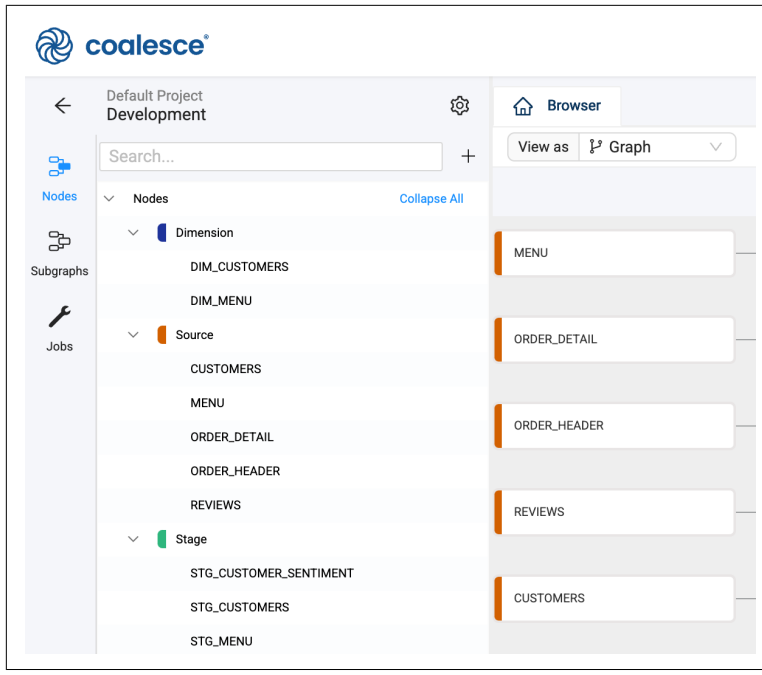


Figure 3-3. The Workspace Settings cog, which allows you to configure your workspace settings.

When it comes to building data pipelines, you'll do this work within the Browser, which is where your Directed Acyclic Graph (DAG) and nodes in your pipeline are displayed, as shown earlier in Figure 3-1.

Now that you're familiar with the essential elements of the Build Interface, it's time to start building data pipelines. First up: data sources.

Adding Data Sources

Data sources are a core component to any data pipeline—after all, you can't build a pipeline without a source. Adding data sources in Coalesce is simple and straight forward. In the upper left corner of the Build Interface, select the plus "+" button, and choose Add Sources. This will open the Data Sources modal, which will display all of the storage locations configured in the Workspace.

Each line item is a Storage Location that is pointing to a database and schema. You can see in Figures 3-4 and 3-5 how the Storage Mappings for each Storage Location correspond to the data sources available in the modal.

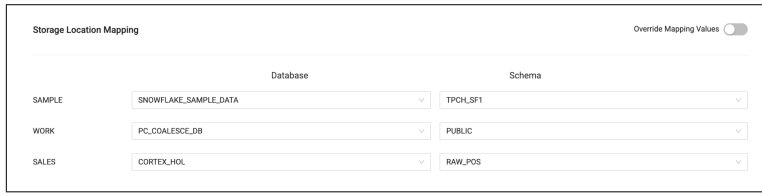


Figure 3-4. storage mappings in the workspace settings, pointing your storage locations to the database and schema in your data platform where your data exists.

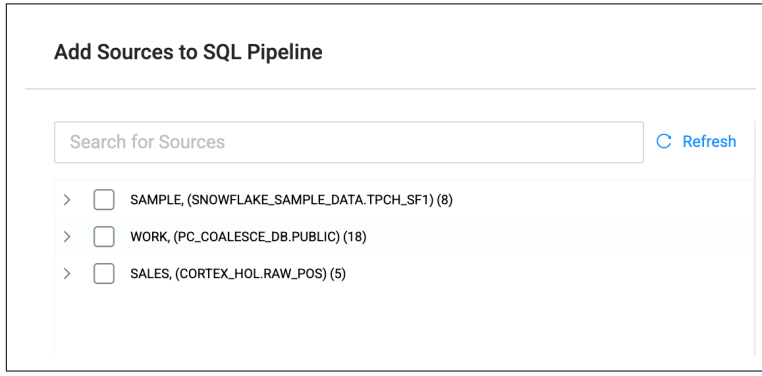


Figure 3-5. The same storage locations and mappings showing up in the data sources selector.

By selecting any of the Storage Locations available, you can view all of the objects available to use within Coalesce. For example, in Figure 3-6, you can see there are eight objects available that we could addinto our pipeline.

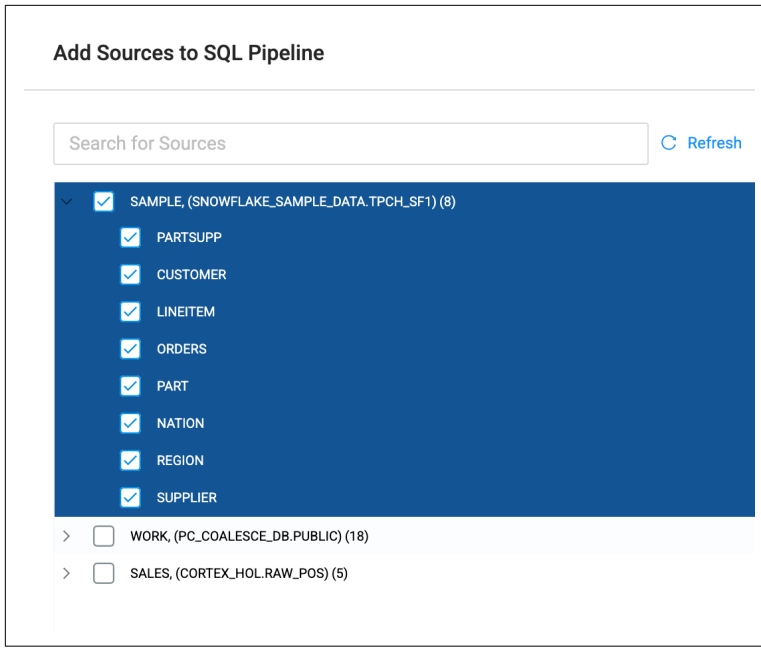


Figure 3-6. Selecting the data sources from the data sources modal.

You can either select the specific objects you need, or select all of them by selecting the checkbox next to the Storage Location name. You can select as many objects between any of the available storage locations. Once you have selected all of the objects you want to add to your pipeline as data sources, select the Add X Sources button in the lower right corner of the modal, where the X represents the number of objects selected.

Coalesce will automatically add all of the objects you selected into the browser of the Build Interface as seen in Figure 3-7. Source Nodes are represented by a dark orange color – as every node has a unique color. This is consistent for any source node added into your Workspace.

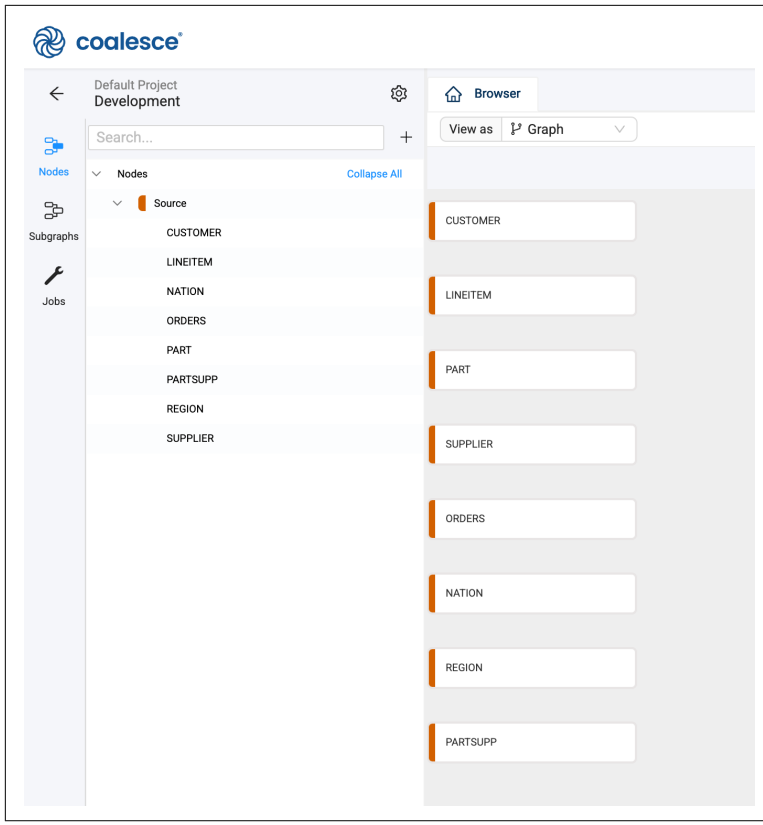


Figure 3-7. Data sources being displayed in the Browser.

With your data sources added into your Workspace, you can now begin transforming this data by using Node Types – whether out of the box, custom node types, or, as we'll see later in this chapter, from Coalesce Marketplace.

Adding Nodes to Your Pipeline

At the core of the developer experience in Coalesce is the ability to quickly add nodes to your data pipeline and start transforming data with ease. Nodes can be added directly from the platform or through Coalesce Marketplace. Because each node follows a standardized structure, the anatomy remains consistent across your pipeline, creating a predictable and unified experience for every developer. Coalesce includes several built-in Node Types to help you move faster right out of the gate:

- Stage
- Persistent Stage
- Dimension
- Fact
- View

You can right click on any of the source nodes and hover over Add Node to view these nodes as shown in Figure 3-8. Let's explore each of the node types below.

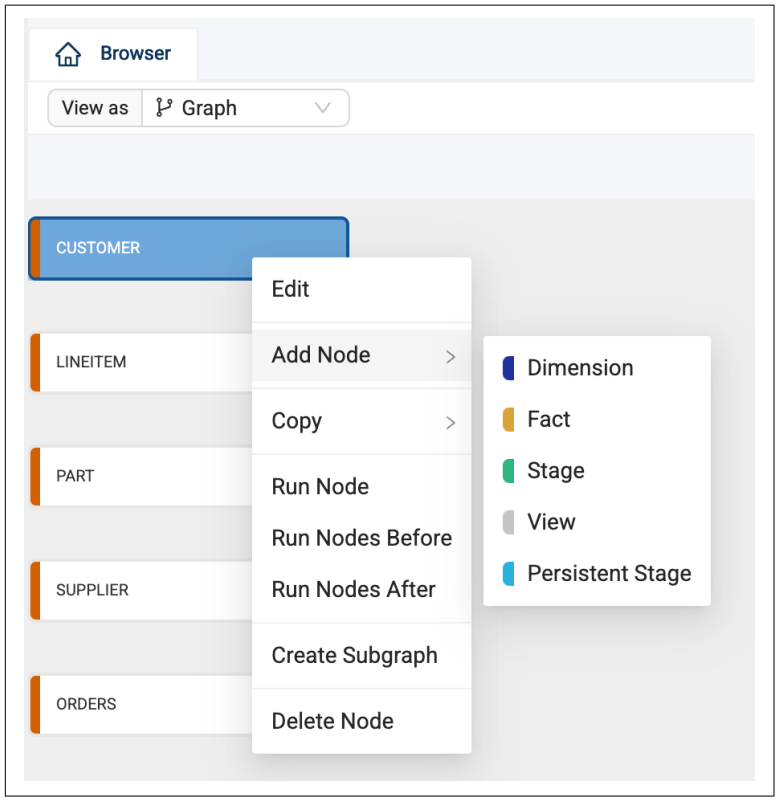


Figure 3-8. Node Types available to add to a data pipeline.

Stage

A Stage Node Type in Coalesce allows you to develop and deploy work in a table or view. It provides an intermediary working or staging layer to store, prepare, and transform raw data before down-

stream tables in your data pipeline use these data. This staging layer is a common data engineering practice for preparing your data. Just as a cook needs to prepare raw ingredients such as carrots and onions by slicing and dicing them before cooking, you can think of Stage Node Types as the preparation layer.

Stage Node Types are by default set to truncate the data in your table during each execution. This means that all data is deleted from the table and the fresh data coming from the upstream node will be processed into the Stage node. If the truncation is deactivated, the execution nature of the node would be to append data to the underlying table.

Persistent Stage

Similar to a Stage, a Persistent Stage is an intermediary Node Type that provides data persistence across execution cycles. Unlike a Stage Node Type, a Persistent Stage contains a business key allowing you to determine the unique identifier for your data source, which, in turn, allows you to persist or store historical data and load net new records into the object. This functionality is particularly beneficial when the objective is to retain historical data for prolonged durations.

Dimension

Coalesce supports Type 1 and Type 2 Slowly Changing Dimensions (SCD) out of the box. Each Dimension requires a business key, or unique identifier, to be defined. You then have the option of defining changing tracking columns, which will automatically configure the object as a Type 2 SCD when columns are selected. This functionality is particularly beneficial in tracking changes to the dimensions (data that describe your facts) in your data.

For example, you may want to know any time your customer's address changes when data is loaded from your sales data source. You can simply select the address column in your data source as a change tracking column, and Coalesce will automatically generate best practice Type 2 SCD Structured Query Language (SQL) within your data platform, saving you hours of developing time.

Fact

Fact Node Types provide you the ability to develop and deploy tables containing the measures or facts in your data platform. Each Fact Node Type contains a business key as well as functionality around varying operations for working with measures i.e. revenue, cost of goods sold, profit, etc. By using Fact Node Types, you can easily distinguish your Fact and Dimension nodes in the Browser when looking at your DAG.

View

By default, the View Node Type is turned off when using Coalesce for the first time. You can enable it by going to the Build Settings and selecting Node Types and turning on the Enable toggle for the View Node Type. This Node Type allows you to create objects as a view in your data platform, which means it is not storing any data, while also providing the flexibility to write custom SQL directly into the SQL editor.

While these five Node Types listed come out-of-the box, Coalesce is not limited to just these Node Types. As we learned in Chapter 2, you have the ability to create your own Node Types, and as we'll see next, use any of the Nodes from Coalesce Marketplace.

Coalesce Marketplace

Coalesce Marketplace provides a wide array of packages that help bring extensibility to your data pipeline projects. These packages are composed of one or more Node Types that help you solve specific problems or provide functionality to accelerate data development as seen in Figure 3-9.

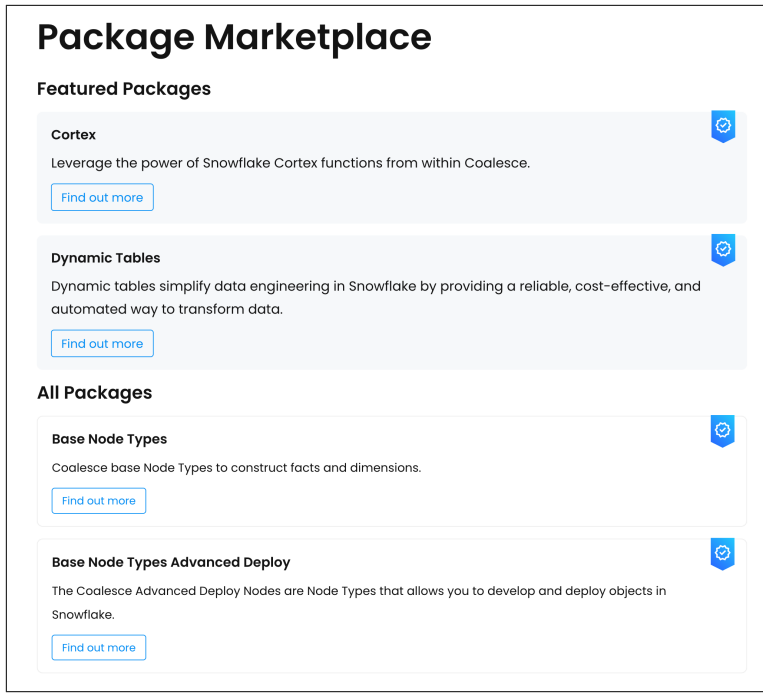


Figure 3-9. Packages available on Coalesce Marketplace that can be added to your Workspace.

Any package with the blue certified checkmark means that Coalesce has certified the package for production environments. Packages that don't include the checkmark are often developed by other engineers or organizations and are not guaranteed to work in all situations. You can see an example of this in Figure 3-10.

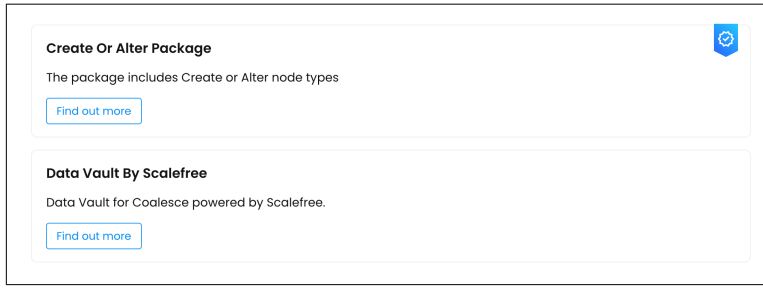


Figure 3-10. A Package that has been certified by Coalesce with the blue checkmark and a package that has not been certified.

You can easily install any of the Node Types from a package by selecting the "Find out more" button on any package. Within the package is a Package ID, and this package ID is how the package is installed within Coalesce.

When inside the Build Settings, you will see the Packages settings. Within the packages settings, you will see the option to either Browse or Install packages, as seen in Figure 3-11. The Browse button will open a new tab and take you to Coalesce Marketplace where you can view all of the packages available to install. The Install button allows you to install a package by providing the Package ID of the package from Coalesce Marketplace.

Figure 3-11. The Browse and Install buttons in the upper right corner of the Packages settings.

When installing a package, you will provide a package ID, version of the package, and an alias. By default, the package will install as the latest version, but you can manually change the version to any previously supported version. The alias of the package is the name or alias under which the Node Types will be displayed in the Browser, as seen in Figure 3-12 and 3-13. Each Node Type that is installed from the package will be displayed in the Node Type settings within the Build Settings of your Workspace. You can then add these Node Types to your pipeline just the same as any other Node Type.

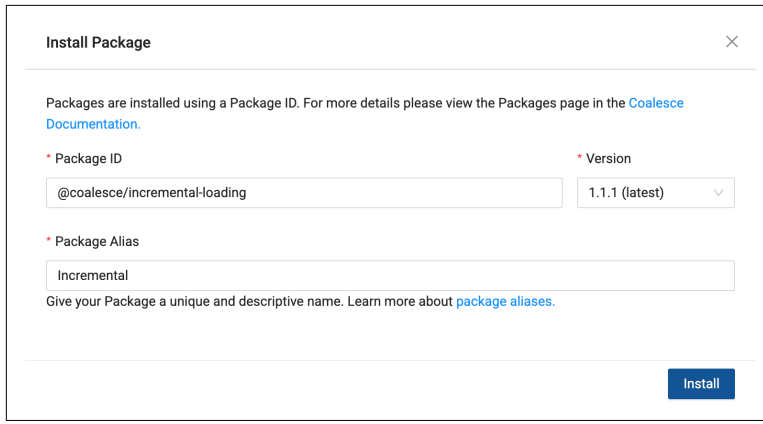


Figure 3-12. Providing an alias when installing a package.

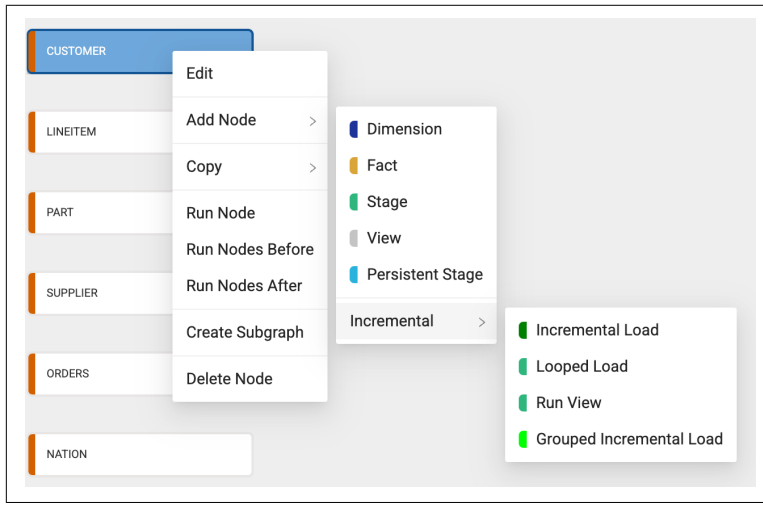


Figure 3-13. How the alias shows up when you add nodes in the Browser.

Whether they are out-of-the box, from Coalesce Marketplace, or custom, you'll use the same method to add all Node Types to your data pipeline. Let's learn how to begin building on our data sources using different Node Types.

Putting Node Types to Work

As we discussed earlier in this chapter, a common pattern of data engineering is to create a staging layer to prepare your data for the

needs of your business users. To do this in coalesce, we can use the Stage Node Type. You can select any node, including source nodes, in Coalesce and right click on the node and hover over Add Node to view all of the Node Types available to add to your data pipeline. In this case, select Stage.

Coalesce will add a green Stage Node Type to the Browser and will automatically add a prefix to the node: STG_, as seen in Figure-Image 3-14. This naming convention provides an additional way to easily see the Stage nodes in your Browser.

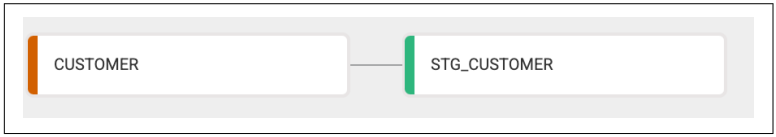


Figure 3-14. The Stage Node Type with the STG_ prefix applied to each node

As you learned about in Chapter 2, Coalesce uses data patterns to provide the templates making up each Node Type. Because each Node Type is a standardized object, you can add them to your pipeline the same way. This also means that you can add them in bulk to multiple objects, since the standard never changes.

In the Browser, you can see multiple nodes selected at once. By right clicking on any of the nodes selected, you can hover over Add Node and, to complete our staging layer, select Stage. Coalesce will automatically add a Stage Node Type to each data source as seen in Figure 3-15, allowing your team to immediately begin transforming your data, without having to configure the code of each object individually.

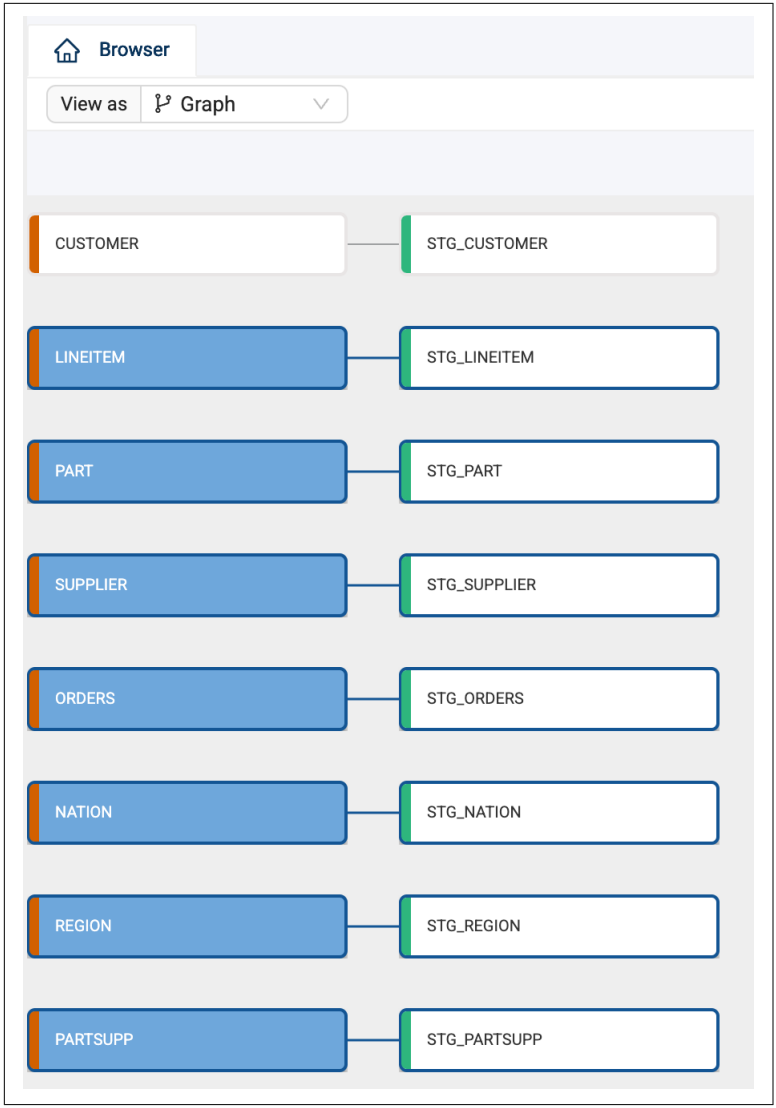


Figure 3-15. Bulk adding Stage nodes to the data pipeline

You can add any other Node Type available in your Workspace the same way that you added Stage Node Types. For instance, you can add a dimension node by right clicking on the STG_CUSTOMER node and selecting Dimension, as shown in Figure 3-16.

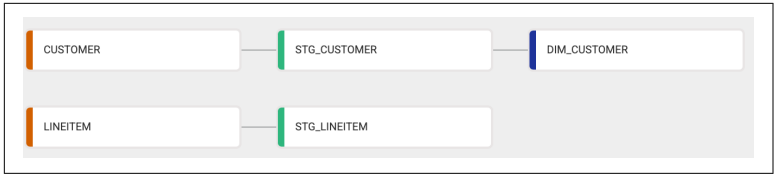


Figure 3-16. STG_CUSTOMER node with a dimension node as a dependency.

You can continue this process for every object needed to develop your data in your pipeline. As you add nodes to your pipeline, you will need to configure them to help you provide solutions to the problems you are solving. In order to do this, you will need to understand the anatomy of a node.

The Anatomy of a Node

In Chapter 2, we discussed how a Node Type is created (Node Definition, Create Template, Run Template). In this chapter, you will learn about the basic anatomy of a node, so you can effectively work with any Node Type. Whenever you double click on a Node in the Browser, it will open the Node Editor. While the configuration options (which you'll learn about shortly) may be different, the basic anatomy of each node is the same. Let's dive into this a bit deeper.

The Mapping Grid

When opening a node for the first time, you'll immediately see the mapping grid, as shown in Figure 3-17. The mapping grid is the display of all of the columns, including the name, data type, transformations, and even comments that are inherited from the parent node(s) in the DAG. You can easily add new columns, apply transformations, change data types, and apply a multitude of other operations in the mapping grid.

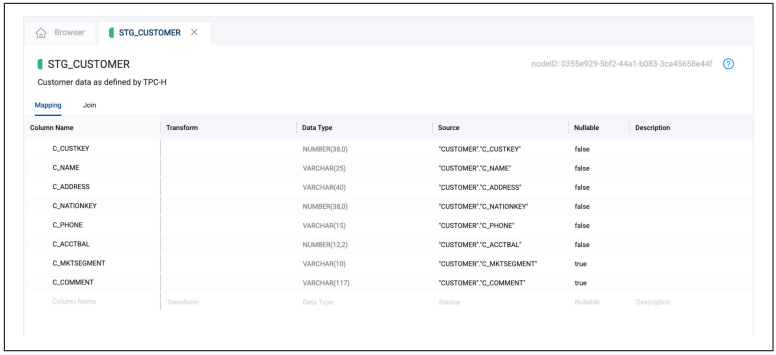


Figure 3-17. The mapping grid of the STG_CUSTOMER node.

Every node in your DAG will contain a mapping grid, even if it only includes a single column or line item, and is your way of knowing which columns you are working with.

The Configuration Options

Each Node Type contains configuration options that are unique to that Node Type. For example, the Dimension Node Type contains configuration options such as a Business Key and Change Tracking columns, which is different from a Stage Node Type which contains other configuration options. The configuration options of each Node Type allow users to accelerate their data development by reusing what has already been created as a standard. This is why configuration of a Type 2 SCD saves hours of time for data developers in Coalesce, because you can configure with just a few clicks, while automatically generating 100s of lines of SQL for you.

You can view the configuration options of any node in the Config tab in the upper right corner of the Node Editor as seen in Figure 3-18.

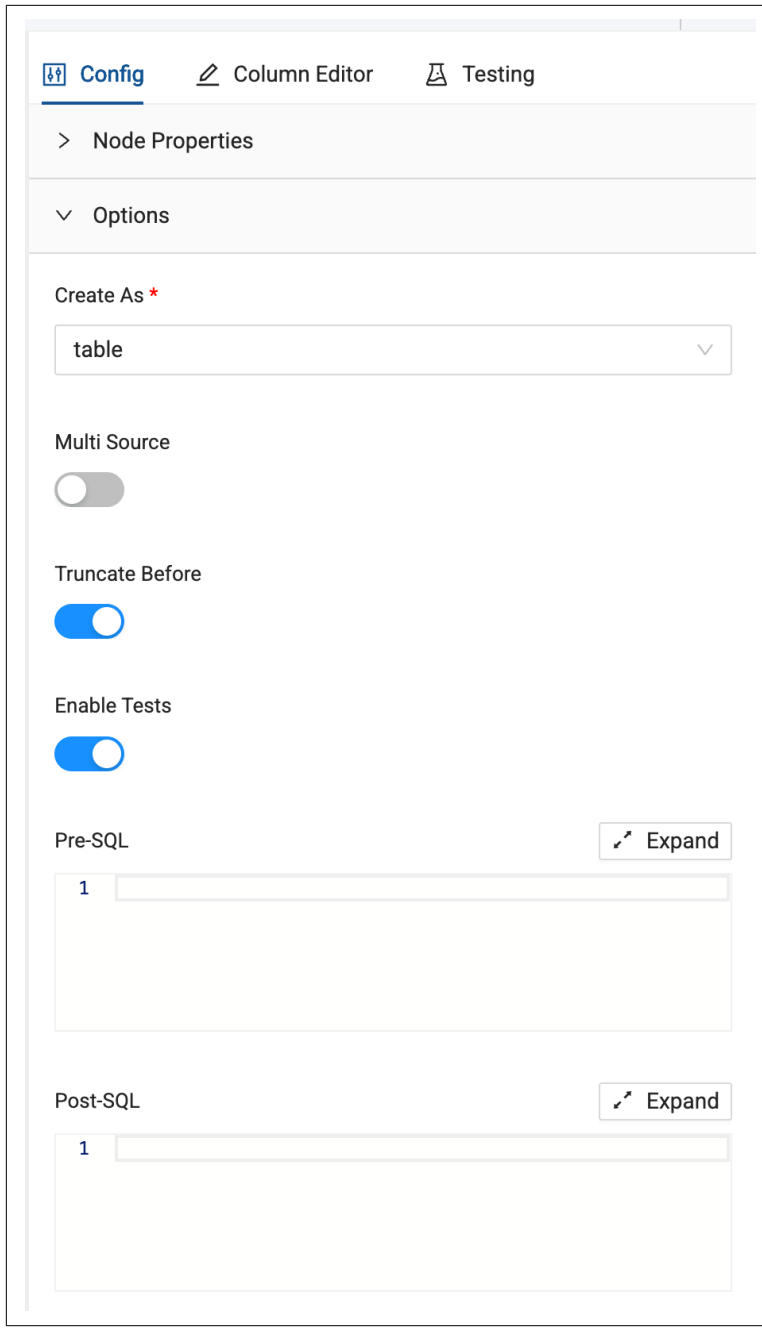


Figure 3-18. The configuration options of a node, providing the ability to configure a node without having to write code.

Create and Run

A critical part of developing data pipelines in Coalesce is the ability to create and run nodes. But what does that mean? When adding a node to your pipeline in Coalesce, the node is not immediately created in your data platform i.e. Snowflake. You as the data developer need to create the object in your data platform. The Create and Run buttons allow you to create objects and then execute any action on the object. Let's break this down in an example.

When working with a Stage node, you have the ability to materialize the object as either a table or a view. Let's assume you choose to build your object as a table. In order for the object to be created in your data platform, you need to select the Create button. In doing so, Coalesce will automatically take all of the metadata about the node, and automatically generate the Data Definition Language (DDL) for the object and create a blank object in your data platform. Coalesce will use the name of the node as the name of the object in your data platform as shown in Figures 3-19 and 3-20.

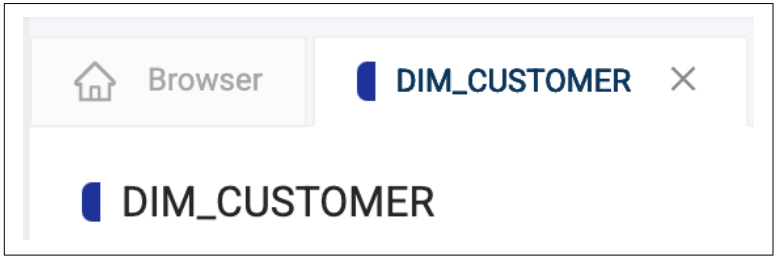


Figure 3-19. Name of the object in Coalesce.

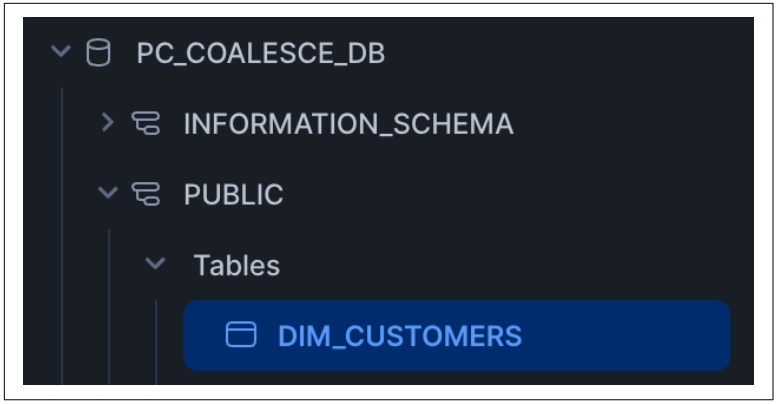


Figure 3-20. The same object created with the same name in your data platform.

Once the Stage is created as a table in your data platform, it is still empty. This is where the Run button comes in. By selecting Run within the node, Coalesce will again take all of the metadata provided from the node (column names, transformations, configurations, etc.), and automatically generate the Data Manipulation Language (DML) for the object, to insert data into the table. You'll be able to see the output of the operation in the Data Preview pane, which will display any data available after a successful run.

This is how Coalesce acts as the interface between your data platform and your raw data. Now you may be thinking that if you need to manually create and run each node, that is not a particularly sustainable approach to pipeline development, and you would be right! In Chapter 4, you will learn about deployments, and how you can automatically schedule your data pipelines to refresh, which effectively runs any of the Create and Run operations as needed.

The Join Tab

Each node type contains a Join Tab. This is located next to the mapping grid, as shown in Figure 3-21. Within the Join Tab, you will notice that there is a line of SQL already in the editor. The SQL shows a FROM statement with a REF function nestled within some curly braces. This line of SQL is creating the dependency to the upstream or parent node i.e. the REFerence to the parent.

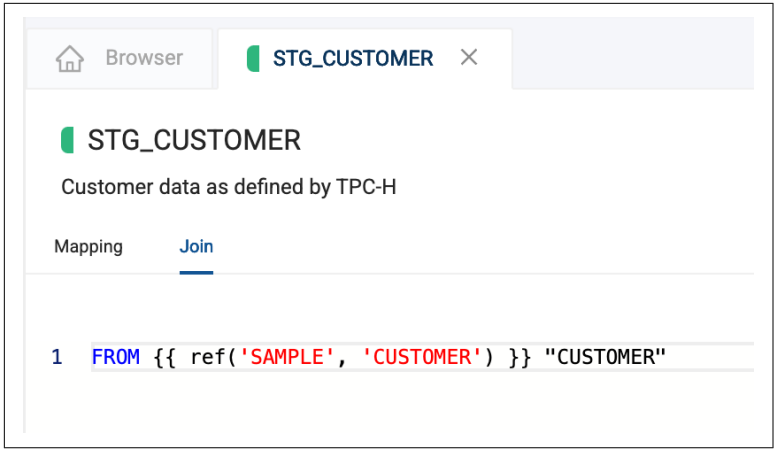


Figure 3-21. The Join Tab next to the mapping grid.

Within the Join Tab, you can perform multiple operations such as joins, filters, and even window functions. The Join Tab is often used for applying transformations to the object as a whole, rather than a singular column. We'll dig into the Join Tab more shortly, but for now, you only need to know where it is and what it can do.

You may notice a few other options within each node, and we'll get to those later in the chapter, but now it's time to get to the core of what nodes are doing, transforming data!

Data Transformations in Coalesce

In this section, I will provide a general overview of core data transformation functionality in Coalesce, but know that there are virtually an unlimited number of ways you can transform your data in Coalesce using various Node Types and settings. To cover the basics, I'll focus on column level and node level transformations.

Column Level Transformations

To kick things off, let's start with column level transformations. These are applied to individual columns within the mapping grid and are written using any valid SQL expression supported by your cloud platform. Let's look at an example.

In the node in Figure 3-22, we are applying an UPPER() function to a variable character column in order to apply consistent text casing across the field.

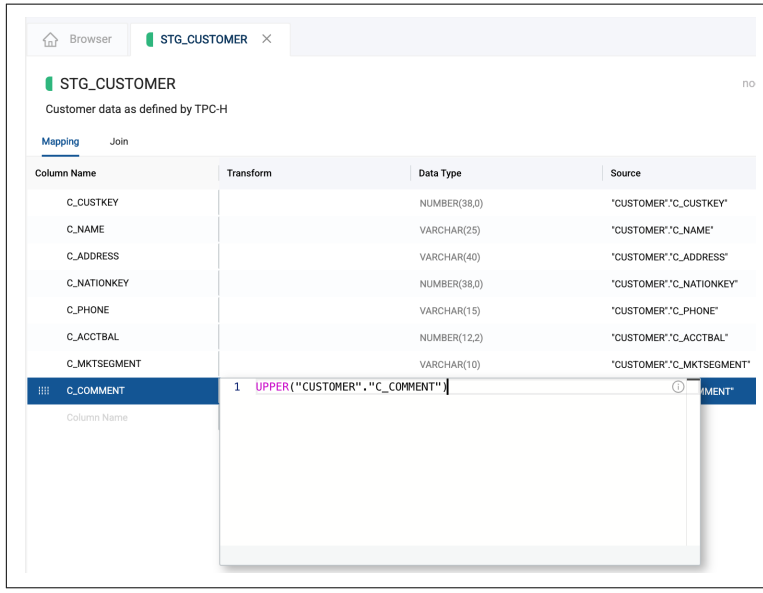


Figure 3-22. Applying a column level transformation to a column in the mapping grid.

While this is a relatively simple example of a column level transformation, you can probably begin to see the ease of management for writing transformations in this way, such as the column level transformations shown within Figure 3-23.

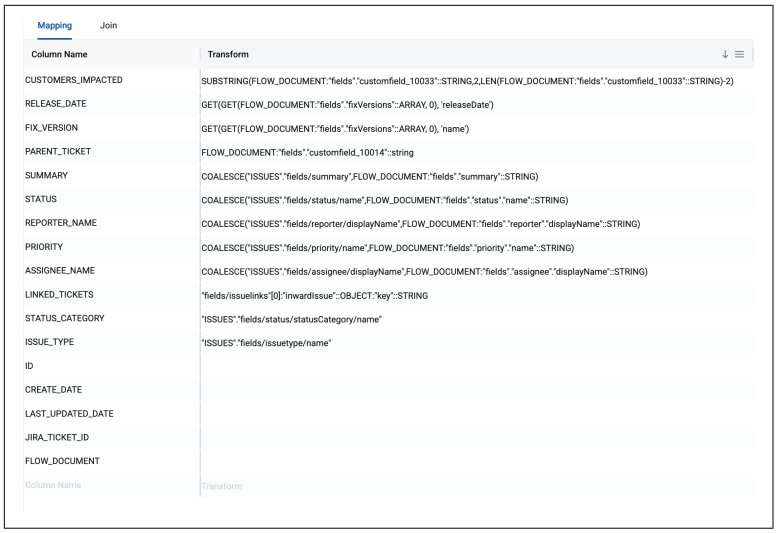


Figure 3-23. Multiple column transformations easily managed in the mapping grid.

Column level transformations can be quite complex, such as nested CASE WHEN statements that contain aggregate functions. Effectively, if you could write the column level transformation in your data platform, you'll be able to write it within Coalesce.

Each column level transformation will contain the name of the column being transformed, as well as the upstream dependency of that column. In this way, you are listing the full table and column reference to provide the metadata to generate the transformation code automatically for you.

It's important to note that column level transformations are only transforming singular columns. You will not be able to filter an entire table in a column level transformation, that is where you would use the Join Tab.

Node Level Transformations

Using the Join Tab, you can apply table or node level transformations to your data. Let's assume you've transformed all of your columns and are ready to apply any SQL necessary to support the logic of the table. For example, maybe you used an aggregate function and now need to supply a GROUP BY—you can supply this in the Join Tab. Maybe you're working with sales data for an

Ecommerce organization and only want to view sales where the order is delivered—you could apply the filter in the Join Tab. You can see examples of these in Figure 3-24. Effectively, if you need to apply any SQL at the table level, you should apply it in the Join Tab.

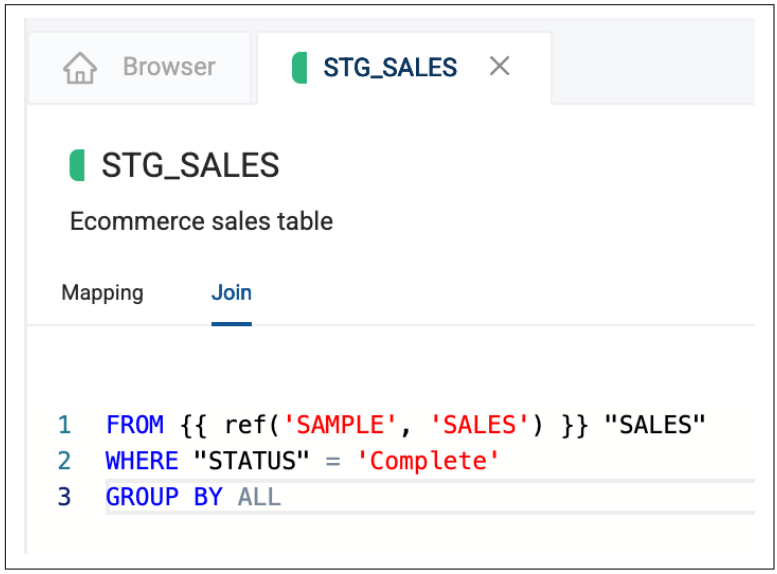


Figure 3-24. Writing SQL to configure the node at the node level.

Whether you are using column level transformations or transforming data at the node level, you can use any SQL supported by your data platform to do so. There is no proprietary syntax limiting your ability to write SQL or use a different language, you can just plug and play. Now that you're familiar with the Join tab, let's explore its namesake–the join itself!

Joins

Up until this point, we have been working with and transforming data in single nodes. But what happens when you want to join multiple nodes together? Coalesce makes this just as easy as adding a node to your data pipeline.

In the Browser, you can select two or more nodes you want to join together. Once selected, you can right click on either node and hover over Join Nodes, and you will see all of the options available for adding a node, but now available as a joined node. In

Figure 3-25, the STG_ORDERS and STG_LINEITEM nodes can be joined together as a new Stage Node Type. Figure 3-26 shows the result of a join in the Browser

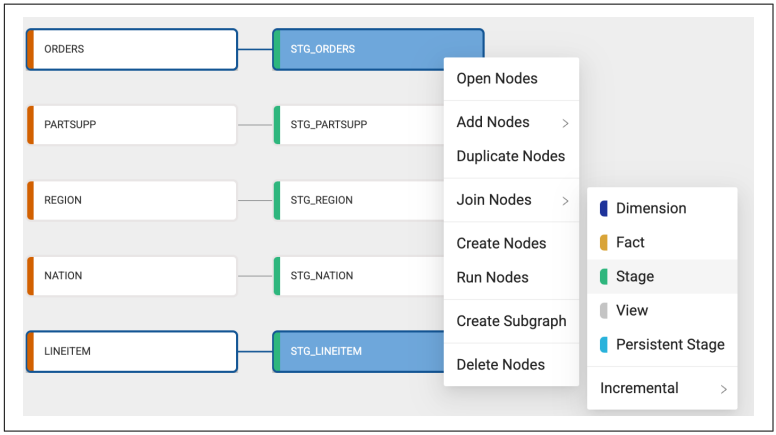


Figure 3-25. Applying a join using the Join Nodes option in the Browser of Coalesce.

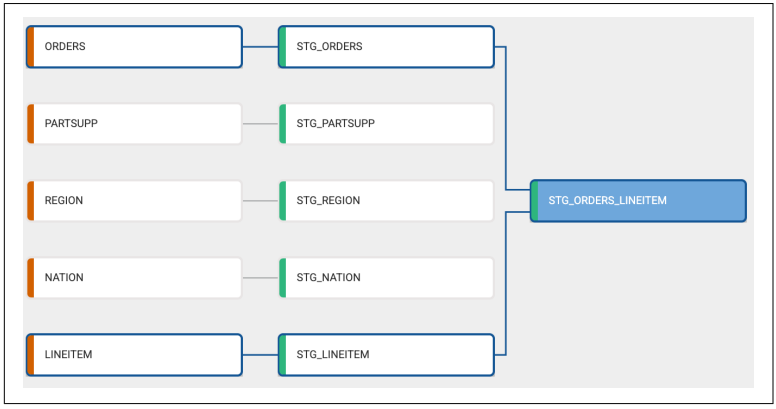


Figure 3-26. The result of the Join Nodes option in the Browser.

Once the joined Stage node is selected, Coalesce will automatically drop you into the Node Editor. To configure the join, navigate to the Join Tab. You will see a join statement that Coalesce has automatically generated for you. In most cases, all you need to do is provide the join condition i.e. the column names used to join the tables together. This can be done by removing the placeholders shown in Figure 3-27 and replacing them with the column names needed to configure the Join as shown in Figure 3-28.

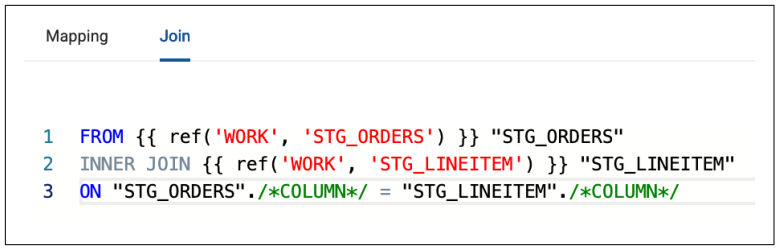


Figure 3-27. Column placeholders in the Join Tab for join conditions.

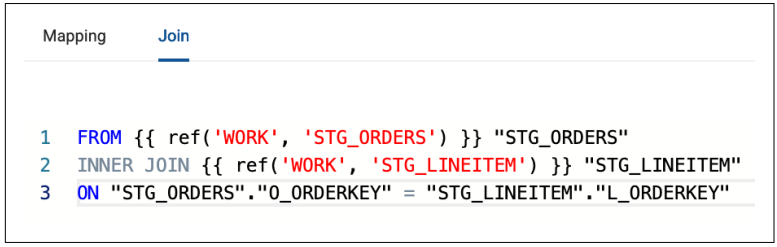


Figure 3-28. Configuration of the join in the table with the proper column condition

As you learned earlier, when it comes to table level transformations, we could always add another join condition if necessary as shown in Figure 3-29.

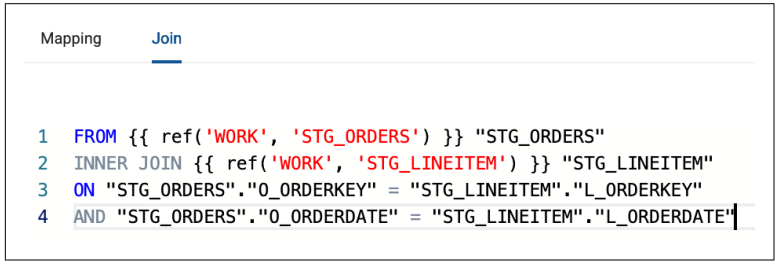


Figure 3-29. Adding another join condition showing flexibility of SQL editor.

Coalesce automatically infers when a join is occurring. If you happen to delete or break your code, you can always have Coalesce regenerate the join. In the upper right corner of the Join Tab SQL editor, you'll see the Generate Join dropdown, as shown in Figure 3-30, which allows you to regenerate the code back in the SQL editor, or copy directly to your clipboard.

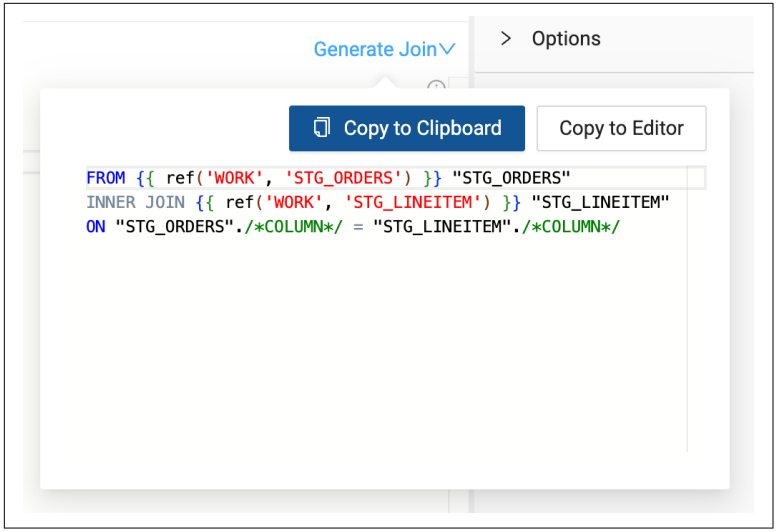


Figure 3-30. The automatic join generation that Coalesce provides in each node.

This same process can be used to join any number of nodes together, making it simple and easy to manage even the most complex joins.

Once tables have been joined, there are often many columns that need additional work—whether it is renaming, changing data types, or applying transformations. In Coalesce, you can handle these tasks efficiently using the bulk editor, which you will learn about next.

Bulk Editing

Joins can cause the need for you to deal with multiple columns at once. In traditional data development scenarios, you may be required to manually delete columns one by one, or apply transformations to one column at a time. But because Coalesce uses data patterns and metadata to accelerate your data development, you can bulk edit columns straight from the mapping grid.

Let's assume you've joined the two nodes together from our previous example. You now have multiple columns containing variable character data types. In the case where you want to apply consistent text casing like our example from earlier, you can bulk add this transformation to each column you select, and only write the transformation one time.

If you were to select all of the VARCHAR columns in the node and right click on any of the columns, you could select Bulk Edit, as shown in Figure 3-31. This would open the Column Editor next to the configuration options you learned about at the beginning of the chapter.

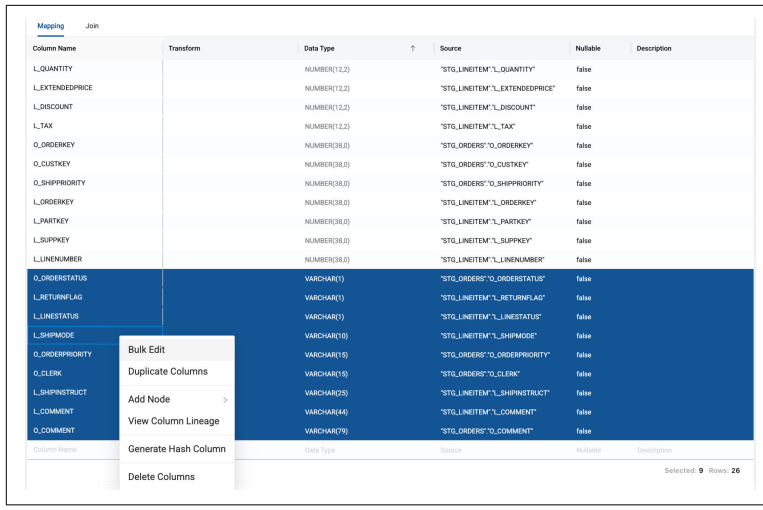


Figure 3-31. FigureImage 3-31. Selecting columns in bulk in order to bulk edit them.

You could then select the attribute you wanted to transform, such as the column names, data types, or transformations. For our example, we'll choose transformation and write a simple UPPER() function that can be applied to each column as shown in Figure 3-32. You'll learn about the token in Chapter 4, but for now know that it automatically resolves the column in any transformation!

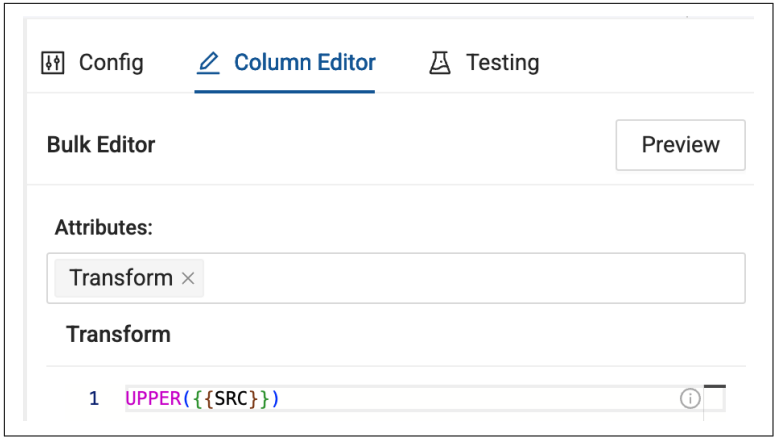


Figure 3-32. The bulk column editor, applying the UPPER function to multiple columns as a transformation.

An attribute of a column can be bulk transformed in this way. Additionally, if we were to want to remove multiple columns, we could easily select all of the columns we wanted to delete, right click on any of the selected columns, and select Delete Columns, as shown in Figure 3-33.

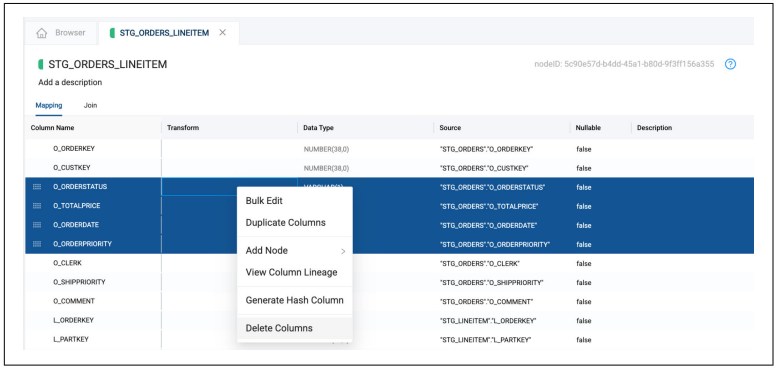


Figure 3-33. Bulk deleting columns from the mapping grid.

By allowing users to work with columns in bulk, Coalesce makes it easy to update multiple fields at once—saving time and letting developers focus on higher-value work.

Data Transformation in Process

There was quite a lot of information to digest in this chapter. You learned about the Build Interface and all of the components it contains. You learned all about adding data sources and how to add Nodes to your pipeline, including the many different kinds of Node Types available to you. You also learned about the anatomy of a node and how to use SQL to transform data inside of any node, while also joining nodes together and applying bulk updates. Whew!

Coalesce is no less powerful because it leads with a user interface. In fact, it enhances efficiency by standardizing the way work is done, ensuring everyone builds from the same foundation. And when you need to write SQL, it is always available. A visual interface does not limit capability—it amplifies it.

In the next chapter, you'll take your new data transformation skills and sharpen them by learning how to manage the data pipelines you build in Coalesce. You're quickly becoming a data developing master.

About the Author

Josh Hall is a data engineer and product marketer that loves to help others learn, explore, and understand the power and impact of the world of data. He spends much of his time creating video content to help others understand technology, writing blogs to distill technical concepts, and traveling to speak at data conferences and user groups. With over half a decade of data consulting experience under his belt, and several more years with data products, Josh wants to take all of his hard earned knowledge and allow others to expedite and amplify their data journey.