REALITY CHECK

While Lakehouse Federation is a powerful tool, it is not a good fit for all use cases. There are some specific examples of use cases when Lakehouse Federation is not a good choice:

- **Real-time data processing:** Lakehouse Federation queries can be slower than queries on data that is stored locally in the lake. Therefore, Lakehouse Federation is not a good choice for applications that require real-time data processing.
- **Complex data transformations:** Where you need complex data transformations and processing, or need to ingest and transform vast amounts of data. For probably the large majority of use cases, you will need to apply some kind of ETL/ELT process against your data to make it fit for consumption by end users. In these scenarios, it is still best to apply a medallion style approach and bring the data in, process it, clean it, then model and serve it so it is performant and fit for consumption by end users.

Therefore, while Lakehouse Federation is a great option for certain use cases as highlighted above, it's not a silver bullet for all scenarios. Consider it an augmentation of your analytics capability that allows for additional use cases that need agility and direct source access for creating a holistic view of your data estate, all controlled through one governance layer.

SETTING UP YOUR FIRST FEDERATED LAKEHOUSE

With that in mind, let's get started on setting up your first federated lakehouse in Databricks using Lakehouse Federation.

For this example, we will be using a familiar sample database — AdventureWorks — running on an Azure SQL Database. We will be walking you through how to set up your connection to Azure SQL and how to add it as a foreign catalog inside Databricks.

PREREQUISITES

To set up Lakehouse Federation in Databricks, you will need the following prerequisites:

- A Unity Catalog-enabled Databricks workspace with Databricks Runtime 13.1 or above and shared or sin...
- A Databricks Unity Catalog metastore
- Network connectivity from your Databricks Runtime cluster or SQL warehouse to the target database systems, including any firewall connectivity requirements, such as here
- The necessary permissions to create connections and foreign catalogs in Databricks Unity Catalog
- The SQL Warehouse must be Pro or Serverless
- For this demo — an example database such as AdventureWorks to use as our data source, along with the necessary credentials. Please see this example.

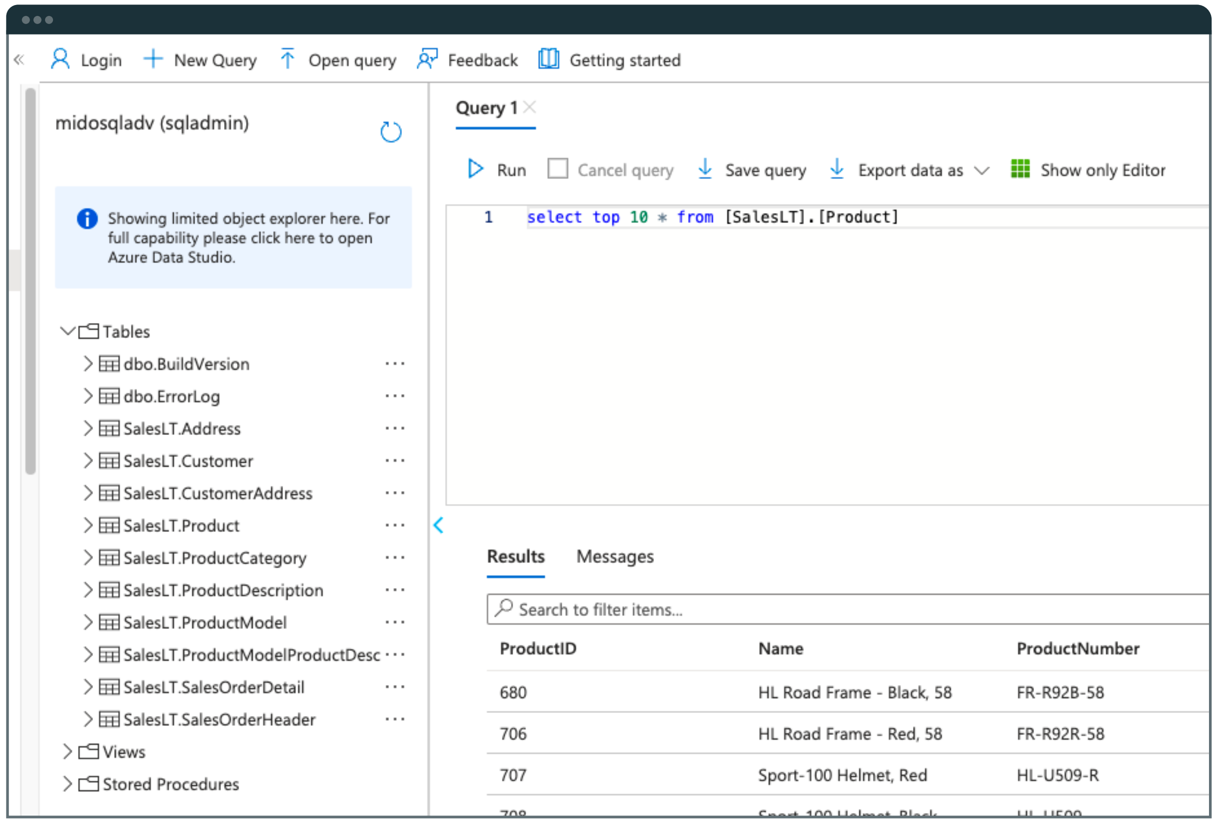
SETUP

Setting up federation is essentially a three-step process, as follows:

- Set up a connection
- Set up a foreign catalog
- Query your data sources

SETTING UP A CONNECTION

We are going to use Azure SQL Database as the test data source with the sample database AdventureWorksLT database already installed and ready to query:


Login | New Query | Open query | Feedback | Getting started

midosqladv (sqladmin)

Showing limited object explorer here. For full capability please click here to open Azure Data Studio.

- Tables
  - dbo.BuildVersion
  - dbo.ErrorLog
  - SalesLT.Address
  - SalesLT.Customer
  - SalesLT.CustomerAddress
  - SalesLT.Product
  - SalesLT.ProductCategory
  - SalesLT.ProductDescription
  - SalesLT.ProductModel
  - SalesLT.ProductModelProductDesc
  - SalesLT.SalesOrderDetail
  - SalesLT.SalesOrderHeader
- Views
- Stored Procedures

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```sql
select top 10 * from [SalesLT].[Product]
```

Results | Messages

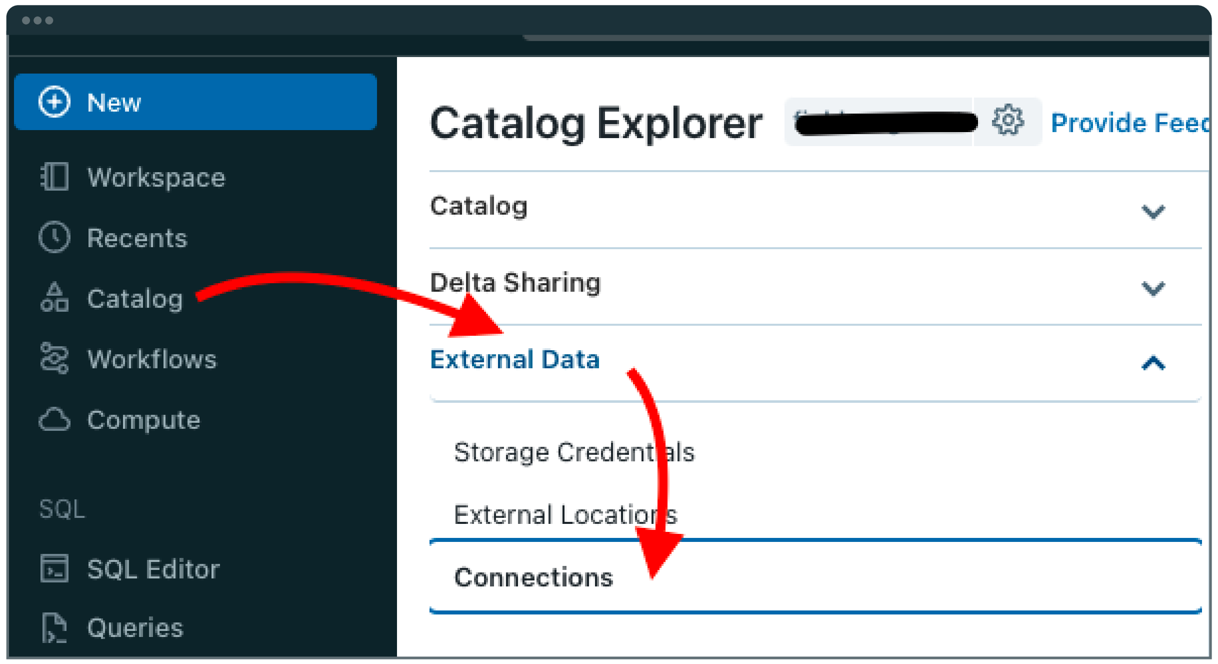
Search to filter items...

ProductID	Name	ProductNumber
680	HL Road Frame - Black, 58	FR-R92B-58
706	HL Road Frame - Red, 58	FR-R92R-58
707	Sport-100 Helmet, Red	HL-U509-R

We want to add this database as a foreign catalog in Databricks to be able to query it alongside other data sources. To connect to the database, we need a username, password and hostname, obtained from my Azure SQL Instance.

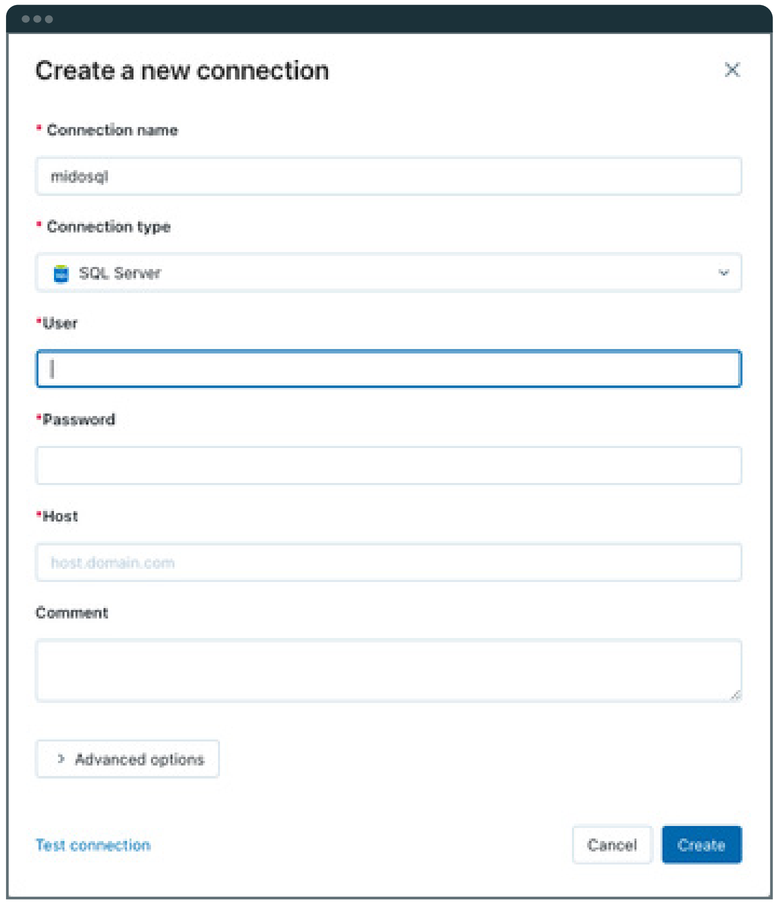
With these details ready, we can now go into Databricks and add the connection there as our first step.

First, expand the Catalog view, go to Connections and click “Create Connection”:



Example query on the source database

To add your new connection, give it a name, choose your connection type and then add the relevant login details for that data source:

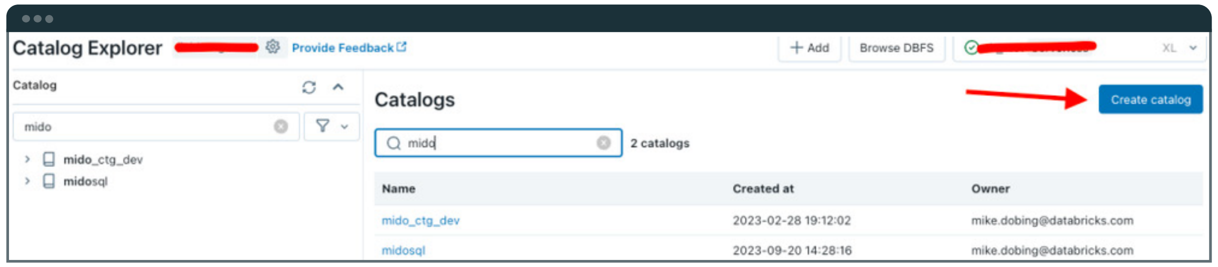


Create a new connection

* Connection name	midosql
* Connection type	SQL Server
*User	
*Password	
*Host	host.domain.com
Comment	

CREATE A FOREIGN CATALOG

Test your connection and verify all is well. From there, go back to the Catalog view and go to Create Catalog:



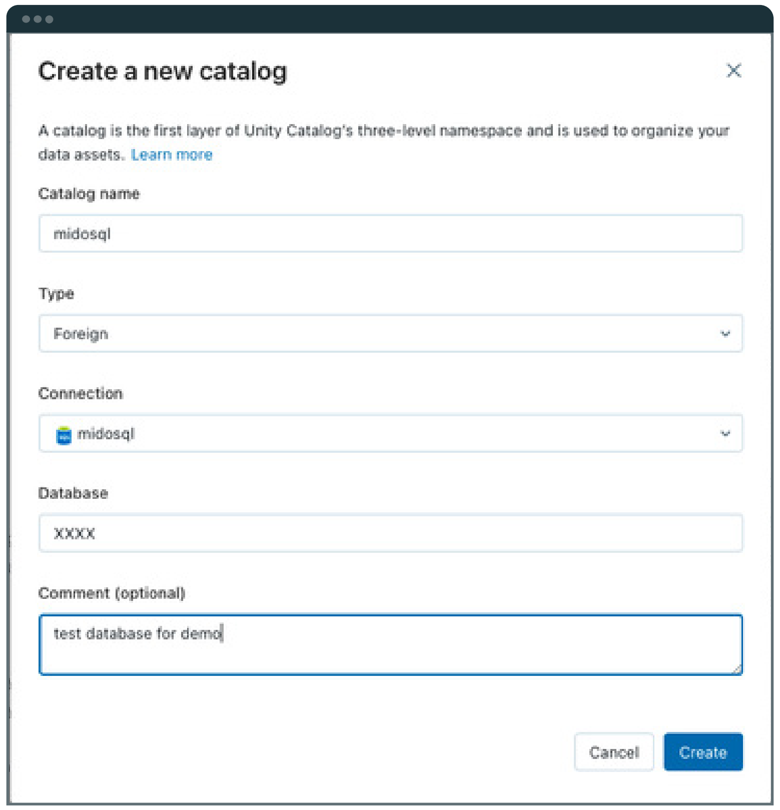
Catalog Explorer [redacted] Provide Feedback

Catalog: mido — mido_ctg_dev, midosql

Catalogs — midd — 2 catalogs

Name	Created at	Owner
mido_ctg_dev	2023-02-28 19:12:02	mike.dobing@databricks.com
midosql	2023-09-20 14:28:16	mike.dobing@databricks.com

From there, populate the relevant details (choosing Type as "Foreign"), including choosing the connection you created in the first step, and specifying the database you want to add as an external catalog:

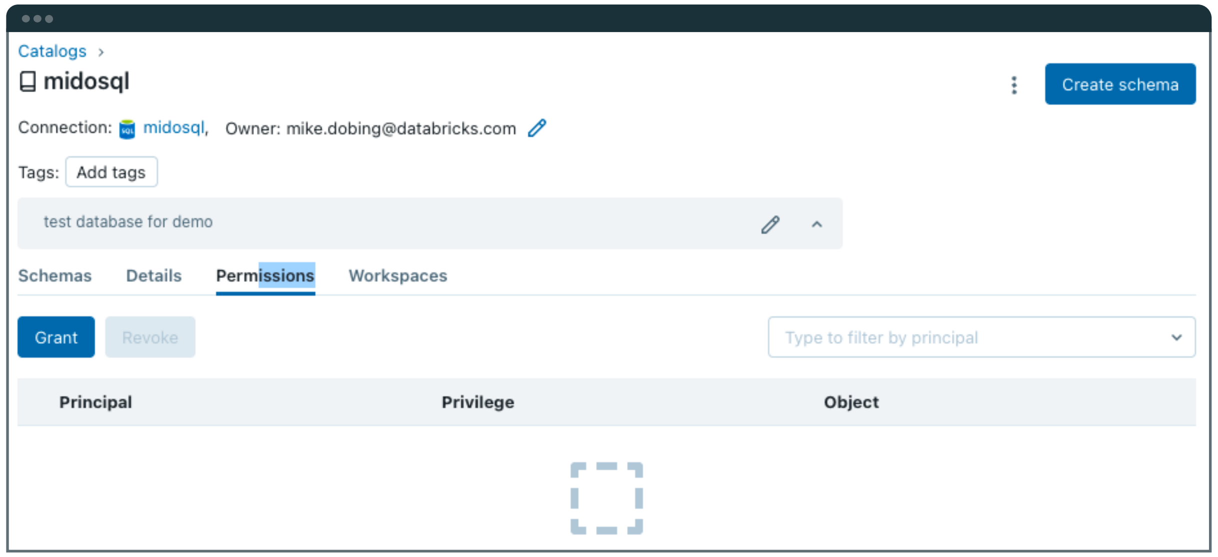


Create a new catalog

A catalog is the first layer of Unity Catalog's three-level namespace and is used to organize your data assets. Learn more

Catalog name	midosql
Type	Foreign
Connection	midosql
Database	XXXX
Comment (optional)	test database for demo

Once added, you can have the option of adding the relevant user permissions to the objects here, all governed by Unity Catalog (skipped this in this article as there are no other users using this database):



Connection: midosql, Owner: mike.dobing@databricks.com

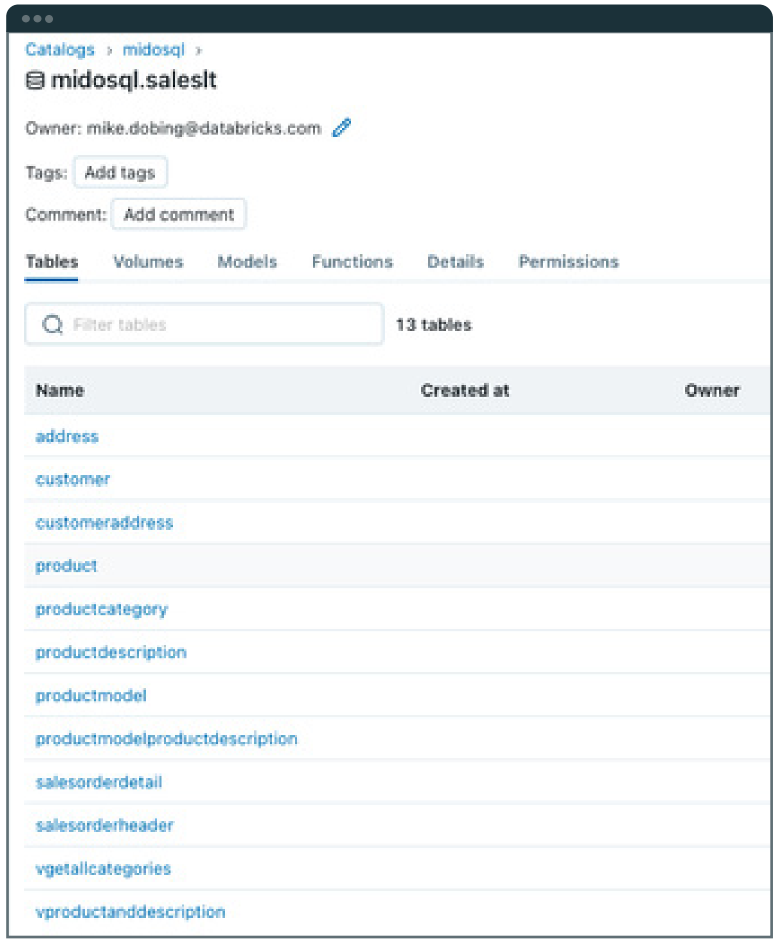
Tags: Add tags

test database for demo

Schemas | Details | Permissions | Workspaces

Principal	Privilege	Object

Our external catalog is now available for querying as you would any other catalog inside Databricks, bringing our broader data estate into our lakehouse:



Owner: mike.dobing@databricks.com

Tags: Add tags

Comment: Add comment

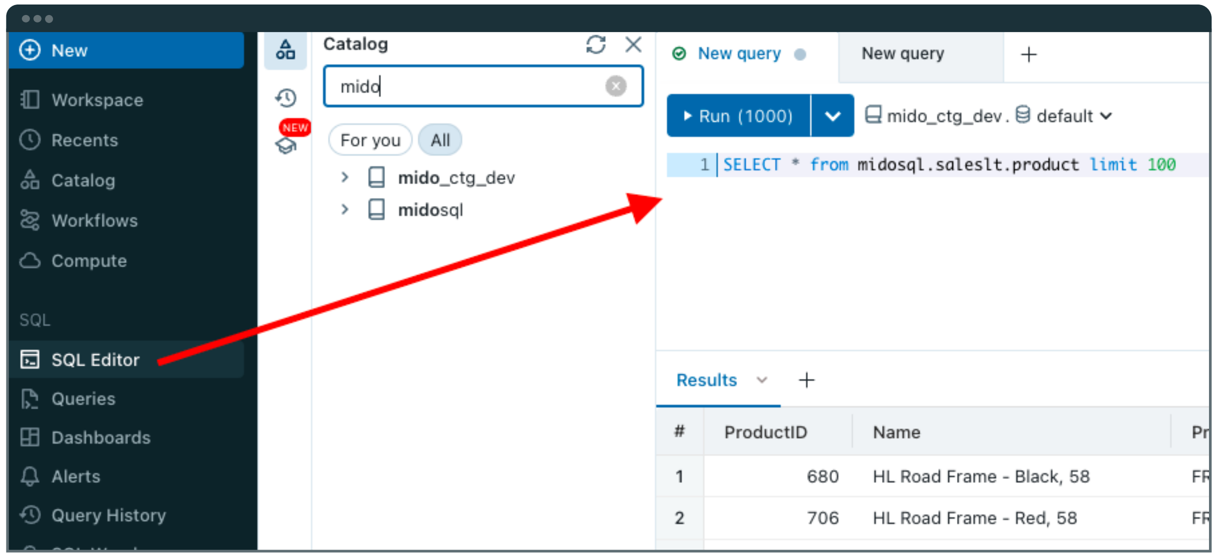
Tables | Volumes | Models | Functions | Details | Permissions

13 tables

Name	Created at	Owner
address		
customer		
customeraddress		
product		
productcategory		
productdescription		
productmodel		
productmodelproductdescription		
salesorderdetail		
salesorderheader		
vgetallcategories		
vproductanddescription		

QUERYING THE FEDERATED DATA

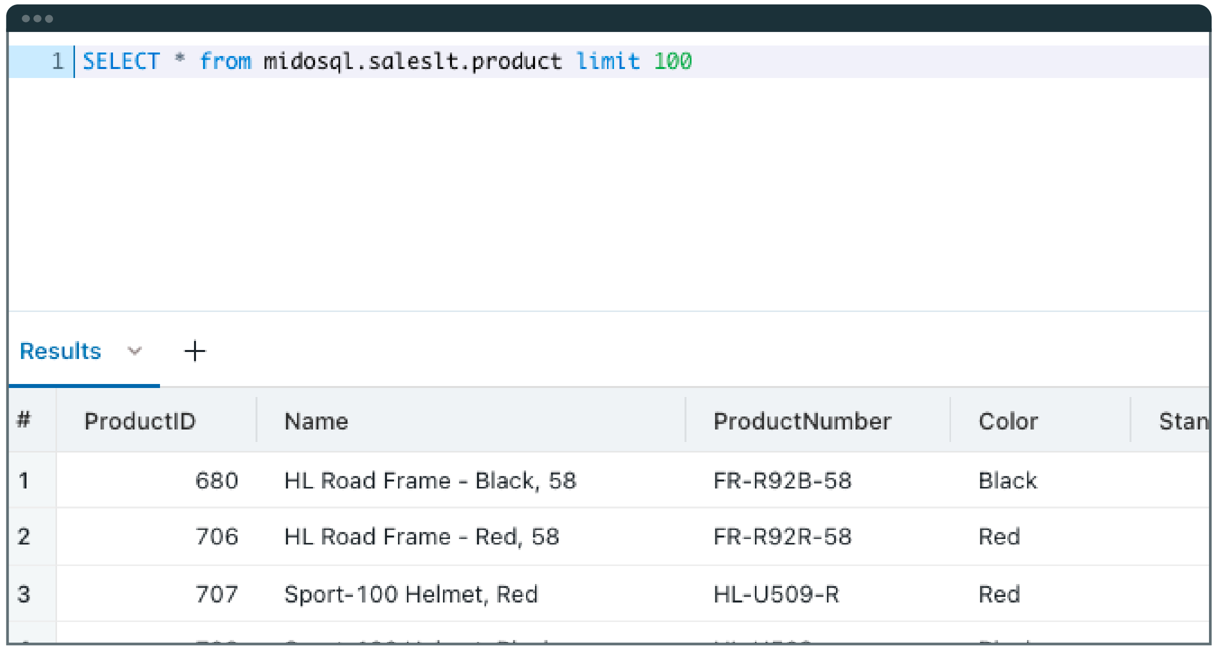
We can now access our federated Azure SQL Database as normal, straight from our Databricks SQL Warehouse:



```
SELECT * from midosql.saleslt.product limit 100
```

#	ProductID	Name	Pr
1	680	HL Road Frame - Black, 58	FR
2	706	HL Road Frame - Red, 58	FR

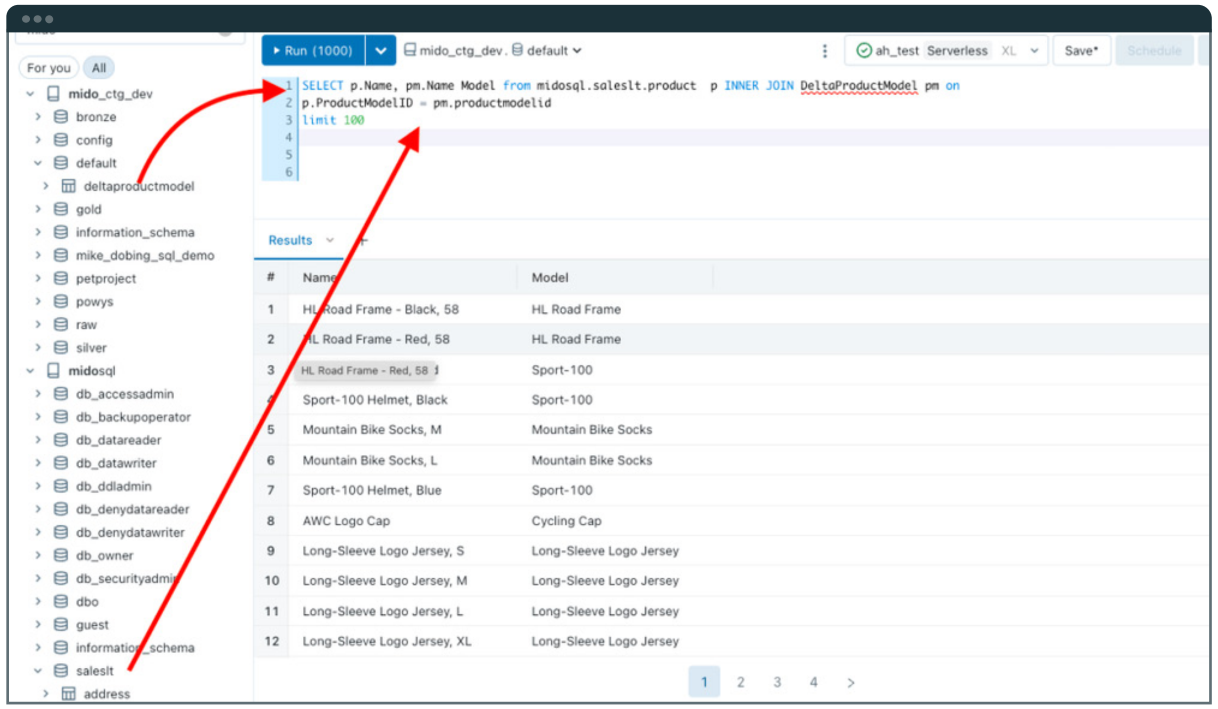
And query it as we would any other object:



```
SELECT * from midosql.saleslt.product limit 100
```

#	ProductID	Name	ProductNumber	Color	Stan
1	680	HL Road Frame - Black, 58	FR-R92B-58	Black	
2	706	HL Road Frame - Red, 58	FR-R92R-58	Red	
3	707	Sport-100 Helmet, Red	HL-U509-R	Red	

Or even join it to a local Delta table inside our Unity Catalog:



```
SELECT p.Name, pm.Name Model from midosql.saleslt.product  p INNER JOIN DeltaProductModel pm on
p.ProductModelID = pm.productmodelid
limit 100
```

#	Name	Model
1	HL Road Frame - Black, 58	HL Road Frame
2	HL Road Frame - Red, 58	HL Road Frame
3	HL Road Frame - Red, 58	Sport-100
4	Sport-100 Helmet, Black	Sport-100
5	Mountain Bike Socks, M	Mountain Bike Socks
6	Mountain Bike Socks, L	Mountain Bike Socks
7	Sport-100 Helmet, Blue	Sport-100
8	AWC Logo Cap	Cycling Cap
9	Long-Sleeve Logo Jersey, S	Long-Sleeve Logo Jersey
10	Long-Sleeve Logo Jersey, M	Long-Sleeve Logo Jersey
11	Long-Sleeve Logo Jersey, L	Long-Sleeve Logo Jersey
12	Long-Sleeve Logo Jersey, XL	Long-Sleeve Logo Jersey

CONCLUSION

What we've shown here is just scratching the surface of what Lakehouse Federation can do with a simple connection and query. By leveraging this offering, combined with the governance and capabilities of Unity Catalog, you can extend the range of your lakehouse estate, ensuring consistent permissions and controls across all of your data sources and thus enabling a plethora of new use cases and opportunities.

Orchestrating Data Analytics With Databricks Workflows

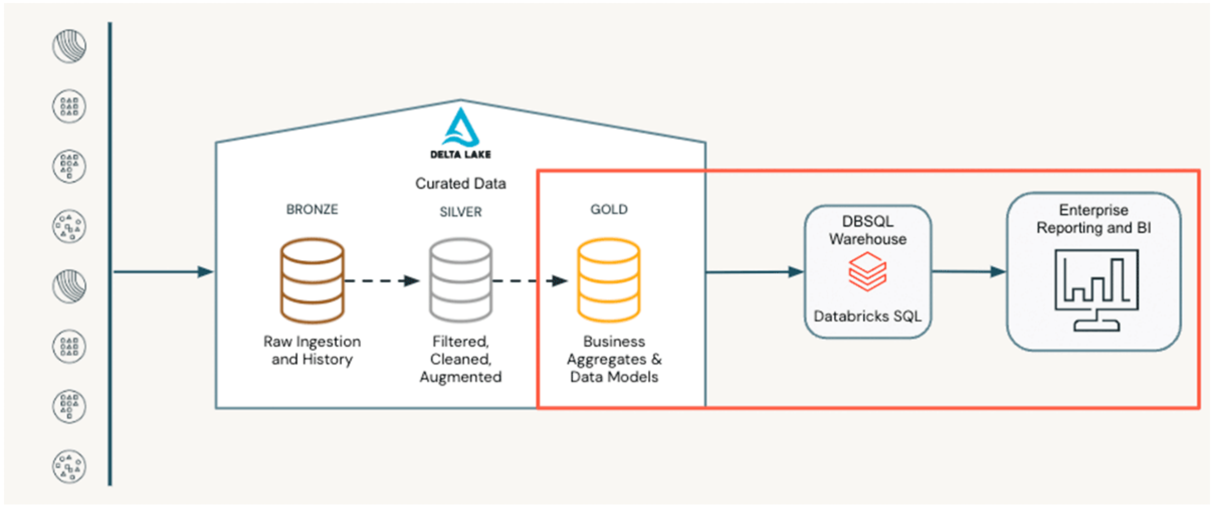
by Matthew Kuehn

For data-driven enterprises, data analysts play a crucial role in extracting insights from data and presenting it in a meaningful way. However, many analysts might not have the familiarity with data orchestration required to automate their workloads for production. While a handful of ad hoc queries can quickly turn around the right data for a last-minute report, data teams must ensure that various processing, transformation and validation tasks are executed reliably and in the right sequence. Without the proper orchestration in place, data teams lose the ability to monitor pipelines, troubleshoot failures and manage dependencies. As a result, sets of ad hoc queries that initially brought quick-hitting value to the business end up becoming long-term headaches for the analysts who built them.

Pipeline automation and orchestration becomes particularly crucial as the scale of data grows and the complexity of pipelines increases. Traditionally, these responsibilities have fallen on data engineers, but as data analysts begin to develop more assets in the lakehouse, orchestration and automation becomes a key piece to the puzzle.

For data analysts, the process of querying and visualizing data should be seamless, and that's where the power of modern tools like Databricks Workflows comes into play. In this chapter, we'll explore how data analysts can leverage Databricks Workflows to automate their data processes, enabling them to focus on what they do best — deriving value from data.

THE DATA ANALYST'S WORLD

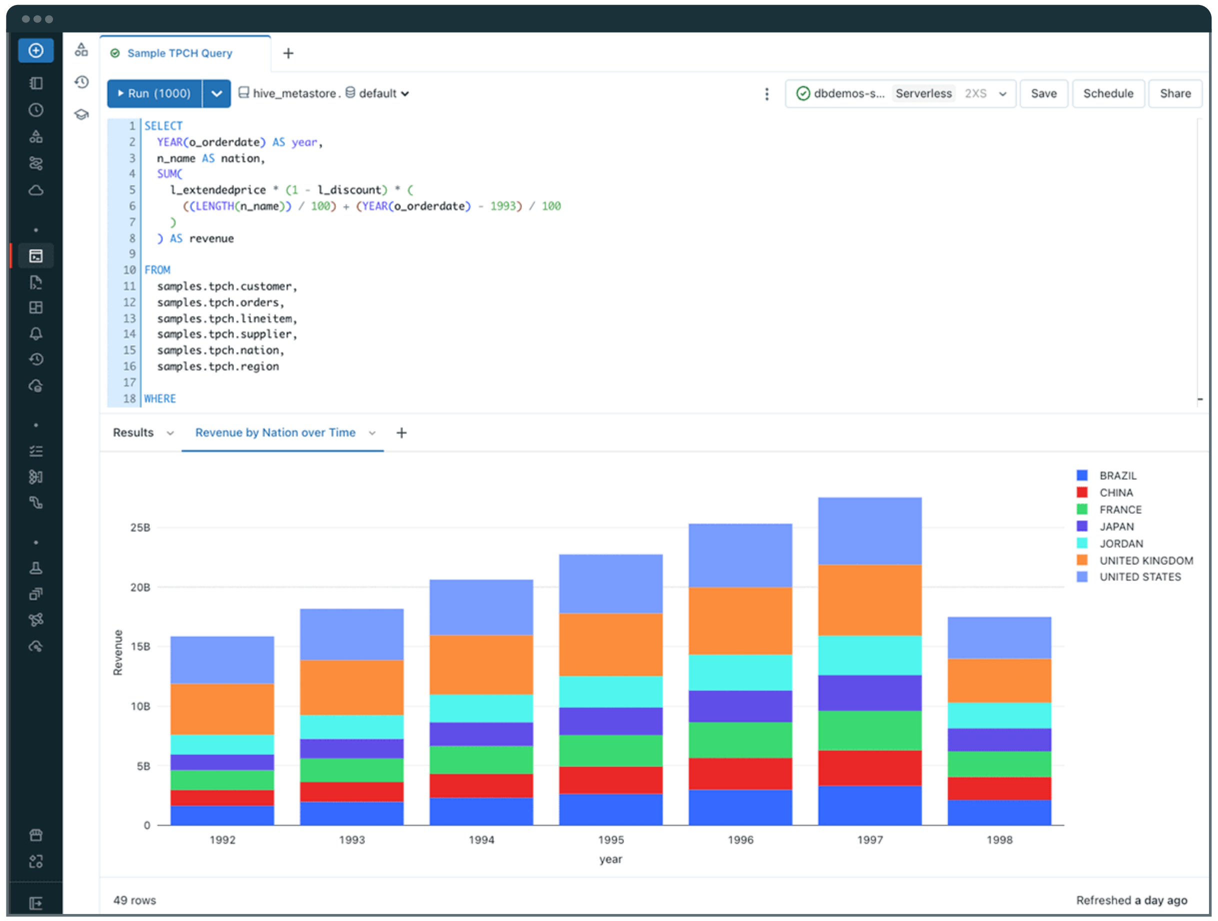

DELTA LAKE
Curated Data
BRONZE
SILVER
GOLD
Raw Ingestion and History
Filtered, Cleaned, Augmented
Business Aggregates & Data Models
DBSQL Warehouse
Databricks SQL
Enterprise Reporting and BI

Data analysts play a vital role in the final stages of the data lifecycle. Positioned at the "last mile," they rely on refined data from upstream pipelines. This could be a table prepared by a data engineer or the output predictions of machine learning models built by data scientists. This refined data, often referred to as the Silver layer in a medallion architecture, serves as the foundation for their work. Data analysts are responsible for aggregating, enriching and shaping this data to answer specific questions for their business, such as:

- "How many orders were placed for each SKU last week?"
- "What was monthly revenue for each store last fiscal year?"
- "Who are our 10 most active users?"

These aggregations and enrichments build out the Gold layer of the medallion architecture. This Gold layer enables easy consumption and reporting for downstream users, typically in a visualization layer. This can take the form of dashboards within Databricks or be seamlessly generated using external tools like Tableau or Power BI via Partner Connect. Regardless of the tech stack, data analysts transform raw data into valuable insights, enabling informed decision-making through structured analysis and visualization techniques.

THE DATA ANALYST'S TOOLKIT ON DATABRICKS



```sql
SELECT
  YEAR(o_orderdate) AS year,
  n_name AS nation,
  SUM(
    l_extendedprice * (1 - l_discount) * (
      ((LENGTH(n_name)) / 100) + (YEAR(o_orderdate) - 1993) / 100
    )
  ) AS revenue

FROM
  samples.tpch.customer,
  samples.tpch.orders,
  samples.tpch.lineitem,
  samples.tpch.supplier,
  samples.tpch.nation,
  samples.tpch.region

WHERE
```

In Databricks, data analysts have a robust toolkit at their fingertips to transform data effectively on the lakehouse. Centered around the Databricks SQL Editor, analysts have a familiar environment for composing ANSI SQL queries, accessing data and exploring table schemas. These queries serve as building blocks for various SQL assets, including visualizations that offer in-line data insights. Dashboards consolidate multiple visualizations, creating a user-friendly interface for comprehensive reporting and data exploration for end users.

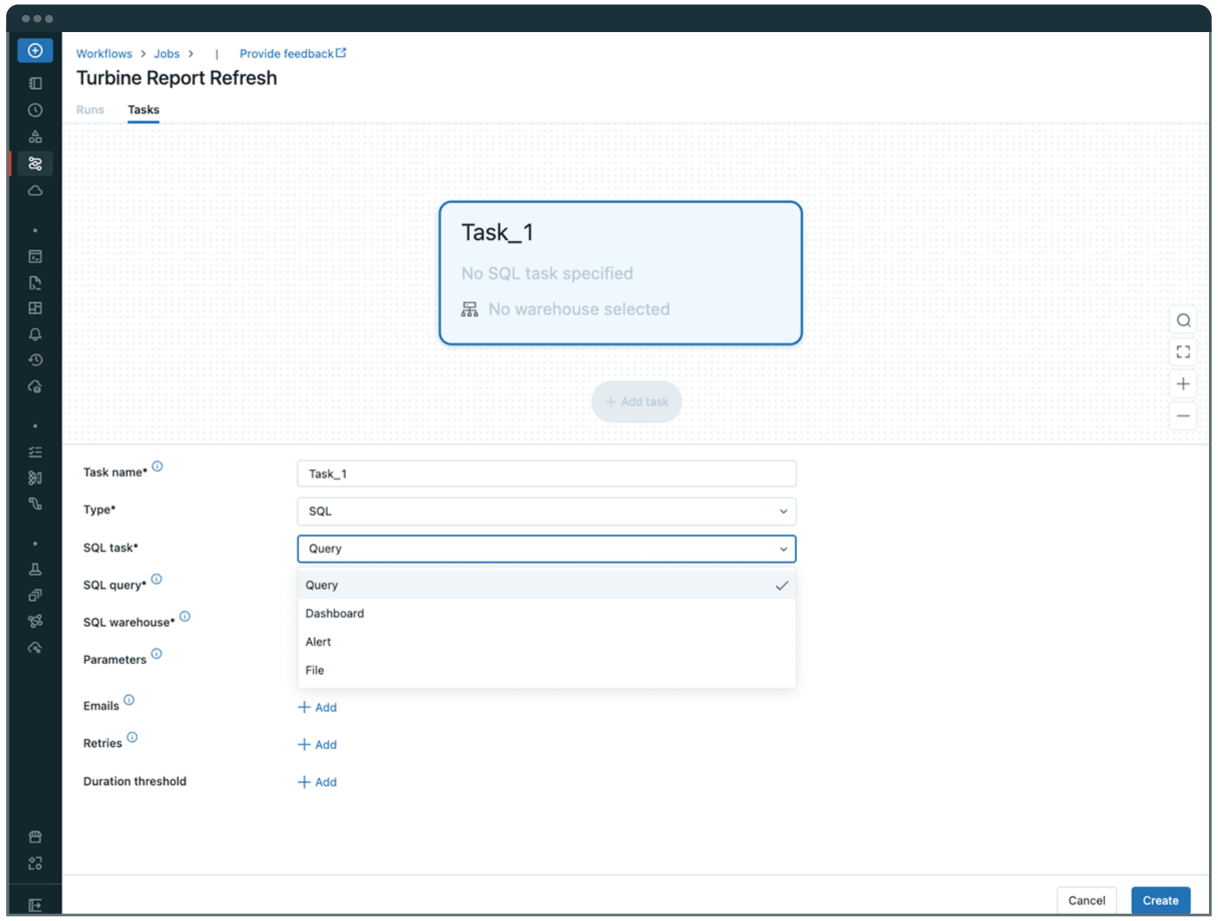
Additionally, alerts keep analysts informed about critical dataset changes in real time. Serverless SQL Warehouses are underpinning all these features, which can scale to handle diverse data volumes and query demands. By default, this compute uses Photon, the high-performance Databricks-native vectorized query engine, and is optimized for high-concurrency SQL workloads. Finally, Unity Catalog allows users to easily govern structured and unstructured data, machine learning models, notebooks, dashboards and files in the lakehouse. This cohesive toolkit empowers data analysts to transform raw data into enriched insights seamlessly within the Databricks environment.

ORCHESTRATING THE DATA ANALYST'S TOOLKIT WITH WORKFLOWS

For those new to Databricks, Workflows orchestrates data processing, machine learning and analytics pipelines in the Data Intelligence Platform. Workflows is a fully managed orchestration service integrated with the Databricks Platform, with high reliability and advanced observability capabilities. This allows all users, regardless of persona or background, to easily orchestrate their workloads in production environments.

Authoring Your SQL Tasks

Building your first workflow as a data analyst is extremely simple. Workflows now seamlessly integrates the core tools used by data analysts — queries, alerts and dashboards — within its framework, enhancing its capabilities through the SQL task type. This allows data analysts to build and work with the tools they are already familiar with and then easily bring them into a Workflow as a Task via the UI.



Task name* Task_1

Type* SQL

SQL task* Query

- Query ✓
- Dashboard
- Alert
- File

SQL query*

SQL warehouse*

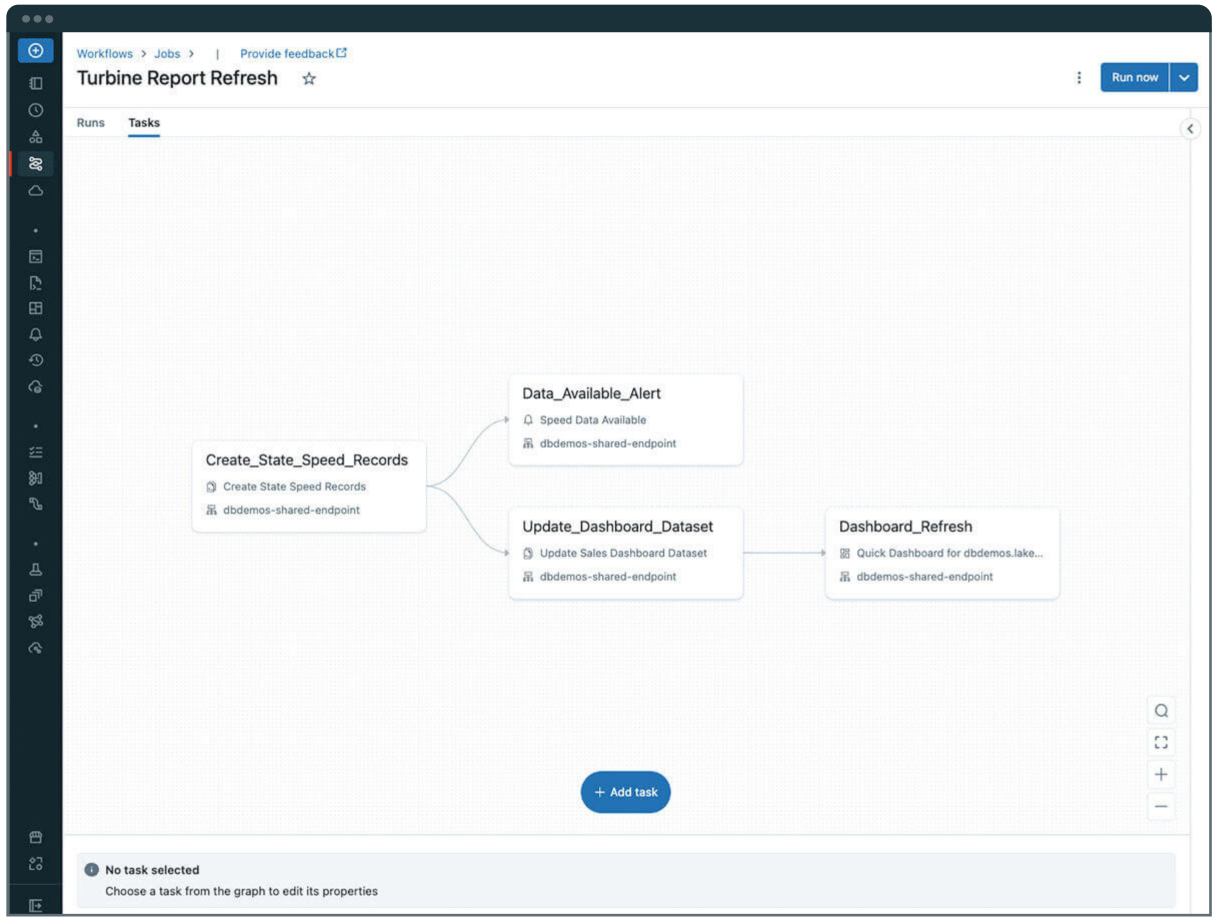
Parameters

Emails + Add

Retries + Add

Duration threshold + Add

As data analysts begin to chain more SQL tasks together, they will begin to easily define dependencies between and gain the ability to schedule and automate SQL-based tasks within Databricks Workflows. In the below example workflow, we see this in action:



databricks

Imagine that we have received upstream data from our data engineering team that allows us to begin our dashboard refresh process. We can define SQL-centric tasks like the ones below to automate our pipeline:

- **Create_State_Speed_Records:** First, we define our refreshed data in our Gold layer with the Query task. This inserts data into a Gold table and then optimizes it for better performance.
- **Data_Available_Alert:** Once this data is inserted, imagine we want to notify other data analysts who consume this table that new records have been added. We can do this by creating an alert which will trigger when we have new records added. This will send an alert to our stakeholder group. You can imagine using an alert in a similar fashion for data quality checks to warn users of stale data, null records or other similar situations. *For more information on creating your first alert, check out this link*.
- **Update_Dashboard_Dataset:** It's worth mentioning that tasks can be defined in parallel if needed. In our example, while our alert is triggering we can also begin refreshing our tailored dataset view that feeds our dashboard in a parallel query.
- **Dashboard_Refresh:** Finally, we create a dashboard task type. Once our dataset is ready to go, this will update all previously defined visualizations with the newest data and notify all subscribers upon successful completion. Users can even pass specific parameters to the dashboard while defining the task, which can help generate a default view of the dashboard depending on the end user's needs.

It is worth noting that this example workflow utilizes queries directly written in the Databricks SQL Editor. A similar pattern can be achieved with SQL code coming from a repository using the File task type. With this task type, users can execute .sql files stored in a Git repository as part of an automated workflow. Each time the pipeline is executed, the latest version from a specific branch will be retrieved and executed.

Although this example is basic, you can begin to see the possibilities of how a data analyst can define dependencies across SQL task types to build a comprehensive analytics pipeline.