

DataOps 03: Trino + DBT + Spark — Everything Everywhere All at Once



Ong Xuan Hong · [Follow](#)

14 min read · Feb 22, 2023

195

2



**Medium**

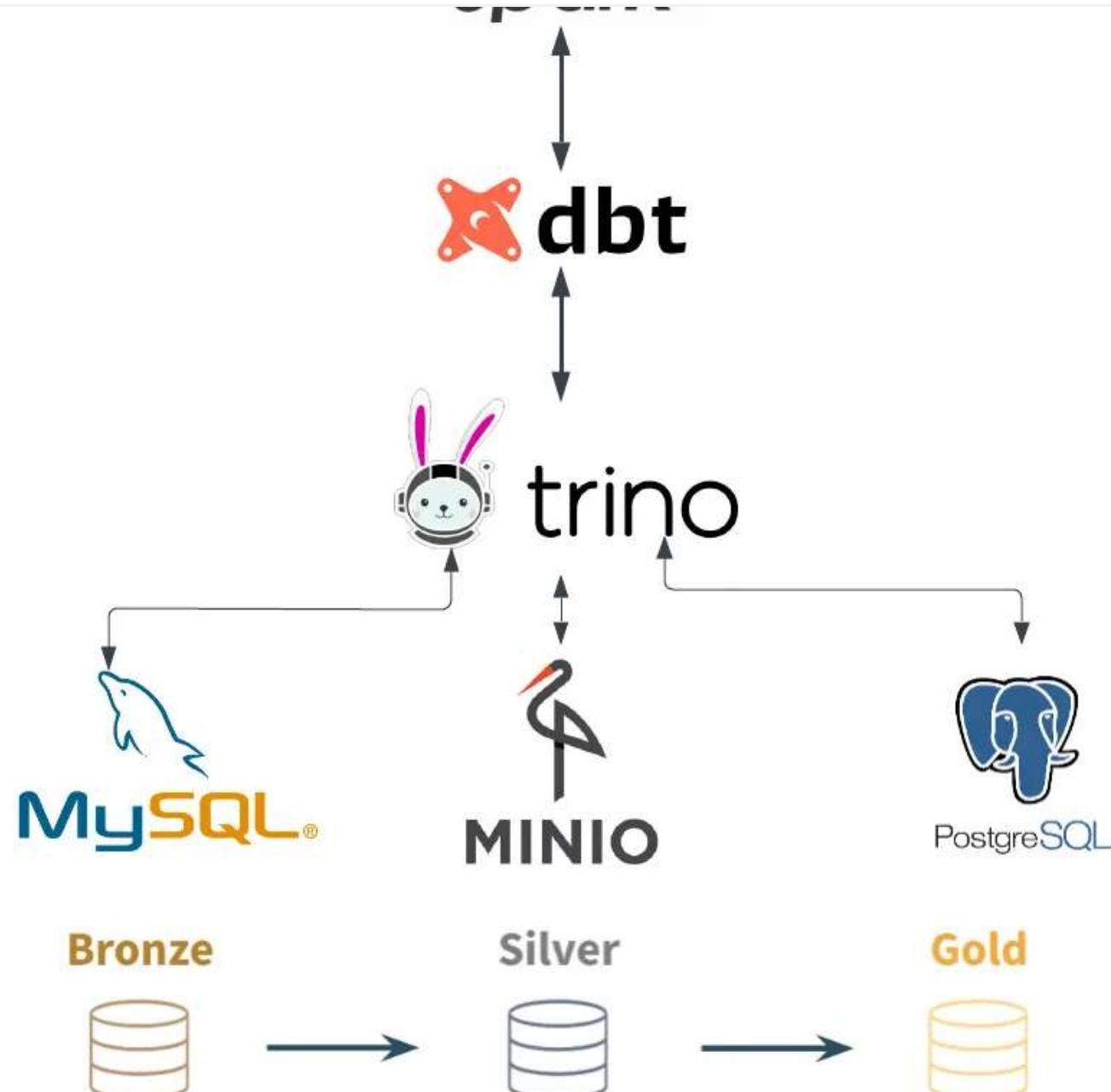
Search



Write

Sign up

Sign in



Raw Ingestion
and History

Filtered, Cleaned,
Augmented

Business-level
Aggregates

I choose a lazy person to do a hard job. Because a lazy person will find an easy way to do it.

— Bill Gates

The inherent human inclination is to conserve energy and embrace moments of rest, a characteristic deeply embedded in our biological programming. Drawing from my experience as a programmer, I have actively explored a diverse array of tools and technologies, including IDEs, frameworks, and no-code platforms, to streamline and reduce the effort required in programming tasks.

In my capacity as a data pipeline creator, I have leveraged tools like pandas to establish connections with various data sources, perform transformations, and load data into output destinations. Apache Spark has also been instrumental in handling similar tasks, significantly expediting the implementation of ETL processes. However, the emergence of a newer technology, known as DBT (Data Build Tool), has notably enhanced the efficiency of analytics engineers.

While I find myself intrigued by the capabilities of no-code platforms like Azure Data Factory, AWS Glue, Airbytes, and Apache Nifi, I sometimes encounter discomfort when it comes to customizing algorithms to align with specific business requirements. To address this challenge, I have discovered a combination of tools that can accelerate ETL workflows while ensuring data quality and adaptability.

After experimenting with various options, I found that using Trino, DBT, and Spark in combination was a perfect solution. **Trino can extract data from everything, DBT can load it everywhere, and Spark can transform it all at once.** In this article, I'll explain how to set up and utilize this powerful combination.

Github: <https://github.com/ongxuanhong/de03-trino-dbt-spark-everything-everywhere-all-at-once>

Data sources: [Kaggle: Brazilian E-Commerce Public Dataset by Olist](#)

Problems



DBT lineage graph

Let's consider a basic analytics problem. Suppose we have a set of data related to product category transactions, and we want to determine the total amount of product sales over a given monthly period, as well as the corresponding values per bill for each product category during these periods. That means we will calculate table `sales_values_by_category` which has the following information

	monthly	category	total_sales	total_bills	values_per_bills
1	2018-08	toys	26,759.44	155	172.642
2	2018-08	telephony	67,187.07	244	275.357
3	2018-08	pet_shop	34,594.29	174	198.818
4	2018-07	toys	22,664.35	150	151.096
5	2018-07	telephony	34,930.6	207	168.747
6	2018-07	pet_shop	27,512.66	138	199.367
7	2018-06	toys	29,234.11	182	160.627
8	2018-06	telephony	31,561.17	205	153.957
9	2018-06	pet_shop	31,558.54	158	199.738
10	2018-05	toys	37,896.28	212	178.756
11	2018-05	telephony	24,947.46	239	104.383
12	2018-05	pet_shop	15,630.43	128	122.113
13	2018-04	toys	29,011.45	193	150.318
14	2018-04	telephony	30,350.14	249	121.888
15	2018-04	pet_shop	25,932.21	155	167.305
16	2018-03	toys	28,692.28	222	129.245
17	2018-03	telephony	32,412.52	322	100.66
18	2018-03	pet_shop	19,144.02	109	175.633
19	2018-02	toys	17,982.83	131	137.274
20	2018-02	telephony	26,698.79	333	80.177

Where:

olist_order_items_dataset

seller_id	shipping_limit_date	price	freight_value
-----	-----	-----	-----

18436dade18ac8b2bce089ec2a04120	2017-09-19 09:45:35	58.9000015259	13.2899999619
d7ddc04e1b6c2c614352b383efe2d36	2017-05-03 11:05:13	239.8999938965	19.9300003052
5b51032eddd242adc84c38acob88f23	2018-01-18 14:48:30	199	17.8700008392
9d7a1d34a5052409006425275ba1c2b	2018-08-15 10:10:18	12.9899997711	12.7899999619
f560393f3a51e74553ab94004ba5c87	2017-02-13 13:57:51	199.8999938965	18.1399993896
5426d21aca402a131fc0a5d0960a3c9	2017-05-23 03:55:27	21.8999996185	12.6899995804
7040e82f899a04d1b434b795a43b461	2017-12-14 12:10:31	19.8999996185	11.8500003815
5996cddab893a4652a15592fb58ab8d	2018-07-10 12:30:45	810	70.75
116b6a846a11724393025641d4edd5e	2018-03-26 18:31:29	145.9499969482	11.6499996185
a143b05f0110f0dc71ad71b4466ce92	2018-07-06 14:10:56	53.9900016785	11.3999996185

olist_order_payments_dataset

order_id	payment_sequential	payment_type	payment_installments
"00010242fe8c5a6d1ba2dd792cb1621	1	credit_card	2
"00018f77f2f0320c557190d7a144bdd	1	credit_card	3
"000229ec398224ef6ca0657da4fc703	1	credit_card	5
"00024acbcdf0a6daa1e931b038114c7	1	credit_card	2
"00042b26cf59d7ce69dfabb4e55b4fd	1	credit_card	3
"00048cc3ae777c65dbb7d2a0634bc1e	1	boleto	1
"00054e8431b9d7675808bcb819fb4a3	1	credit_card	1
"000576fe39319847cbb9d288c5617fa	1	credit_card	10
"0005a1a1728c9d785b8e2b08b904576	1	credit_card	3
"0005f50442cb953dcd1d21e1fb92349	1	credit_card	1

olist_orders_dataset

order_id	customer_id	order_status
"00010242fe8c5a6d1ba2dd792cb1621	"3ce436f183e68e07877b285a838db11	delivered
"00018f77f2f0320c557190d7a144bdd	f6dd3ec061db4e3987629fe6b26e5cce	delivered
"000229ec398224ef6ca0657da4fc703	"6489ae5e4333f3693df5ad4372dab6d	delivered
"00024acbcdf0a6daa1e931b038114c7	d4eb9395c8c0431ee92fce09860c5a06	delivered
"00042b26cf59d7ce69dfabb4e55b4fd	"58dbd0b2d70206bf40e62cd34e84d79	delivered
"00048cc3ae777c65dbb7d2a0634bc1e	"816cbea969fe5b689b39cf97a50674	delivered
"00054e8431b9d7675808bcb819fb4a3	"32e2e6ab09e778d99bf2e0ecd489871	delivered
"000576fe39319847ccb9d288c5617fa	"9ed5e522dd9dd85b4af4a077526d811	delivered
"0005a1a1728c9d785b8e2b08b904576	"16150771dfd4776261284213b89c304	delivered
"0005f50442cb953dc1d21e1fb92349	"351d3cb2cee3c7fd0af6616c82df21d	delivered

olist_products_dataset

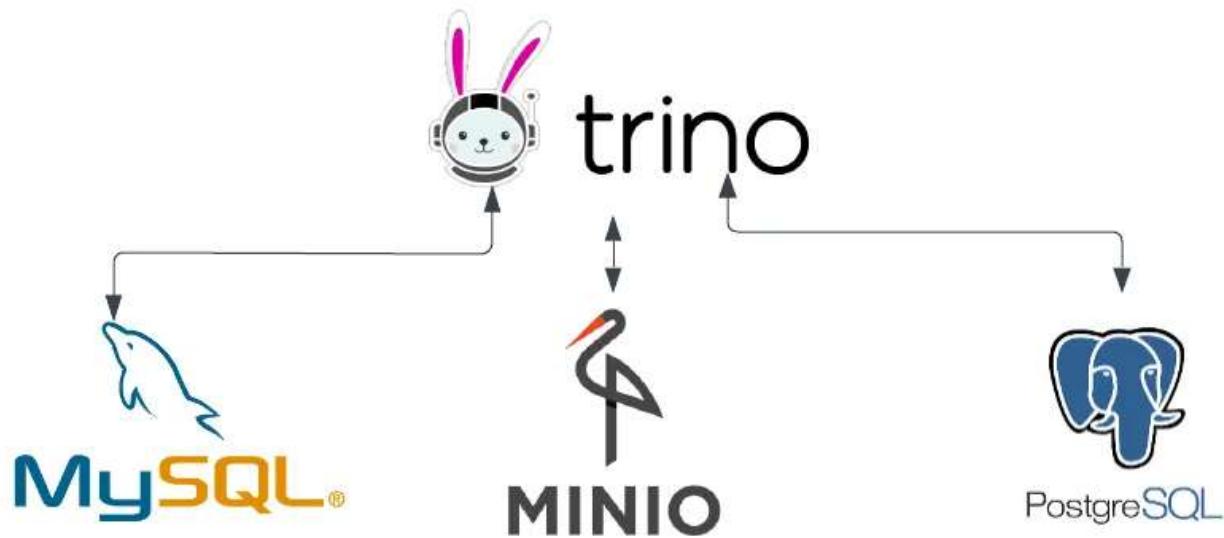
product_id	product_category_name	product_name_length	prod
"00066f42aeeb9f3007548bb9d3f33c3	perfumaria	53	596
"00088930e925c41fd95ebfe695fd265	automotivo	56	752
"0009406fd7479715e4bef61dd91f246	cama_mesa_banho	50	266
"000b8f95fcb9e0096488278317764d1	utilidades_domesticas	25	364
"000d9be29b5207b54e86aa1b1ac5487	relogios_presentes	48	613
"0011c512eb256aa0dbbb544d8dffcf6	automotivo	58	177

"00126f27c813603687e6ce486d909d0 cool_stuff	42	2,46
"001795ec6f1b187d37335e1c4704762 consoles_games	53	274
"001b237c0e9bb435f2e54071129237e cama_mesa_banho	42	253
"001b72dfd63e9833e8c02742adf472e moveis_decoracao	45	520

product_category_name_translation

product_category_name	product_category_name_english
----- -----	
agro_industria_e_comercio	agro_industry_and_commerce
alimentos	food
alimentos_bebidas	food_drink
artes	art
artes_e_artesanato	arts_and_craftmanship
artigos_de_festas	party_supplies
artigos_de_natal	christmas_supplies
audio	audio
automotivo	auto
bebes	baby

Trino — the EXTRACTOR of everything



Normally, when working with Pandas, in order to connect to a data source, you need to create DataLoader class for handling the connection. For example, to connect to MySQL you will define a function similar to the script below:

```
def get_db_connection(self):
    conn_info = (
        f"mysql+pymysql://{{self.params['user']}:{{self.params['password']}}}"
        + f"@{{self.params['host']}:{{self.params['port']}}}"
        + f"/{{self.params['database']}}")
```

```
print(f"Configs: {conn_info}")
db_conn = create_engine(conn_info).connect()
try:
    yield db_conn
except Exception:
    raise
finally:
    db_conn.close()
```

If you'd like to connect to PostgreSQL, you might create a function similar to this script

```
def get_db_connection(self):
    conn_info = (
        f"postgresql+psycopg2://{{self.params['user']}:{{self.params['password']}}}"
        + f"@{{self.params['host']}:{{self.params['port']}}}"
        + f"/{{self.params['database']}}"
    )
    print(f"Configs: {conn_info}")
    db_conn = create_engine(conn_info)
    return db_conn
```

Or MinIO, the connection would be like this

```
def get_db_connection(self):  
  
    client = Minio(  
        endpoint=self.params.get("endpoint_url"),  
        access_key=self.params.get("aws_access_key_id"),  
        secret_key=self.params.get("aws_secret_access_key"),  
        secure=False,  
    )  
  
    try:  
        yield client  
    except Exception:  
        raise
```

I'm wondering if there is a framework that can simplify the process of connecting to various data sources through a simple configuration rather than requiring manual coding. And Trino is a good candidate because it supports **federated queries** with the ability to access data from multiple systems using a single query, it becomes feasible to combine historical log data from an S3 object storage with customer data from a MySQL relational database.

A list of Trino connectors can be [found here](#). In our demo, we will use the following connectors:

- Delta Lake

```
connector.name=delta-lake
hive.metastore.uri=thrift://hive-metastore:9083
hive.s3.endpoint=http://minio:9000
hive.s3.aws-access-key=minio
hive.s3.aws-secret-key=minio123
hive.s3.path-style-access=true
delta.enable-non-concurrent-writes=true
delta.unique-table-location=false
```

- MySQL

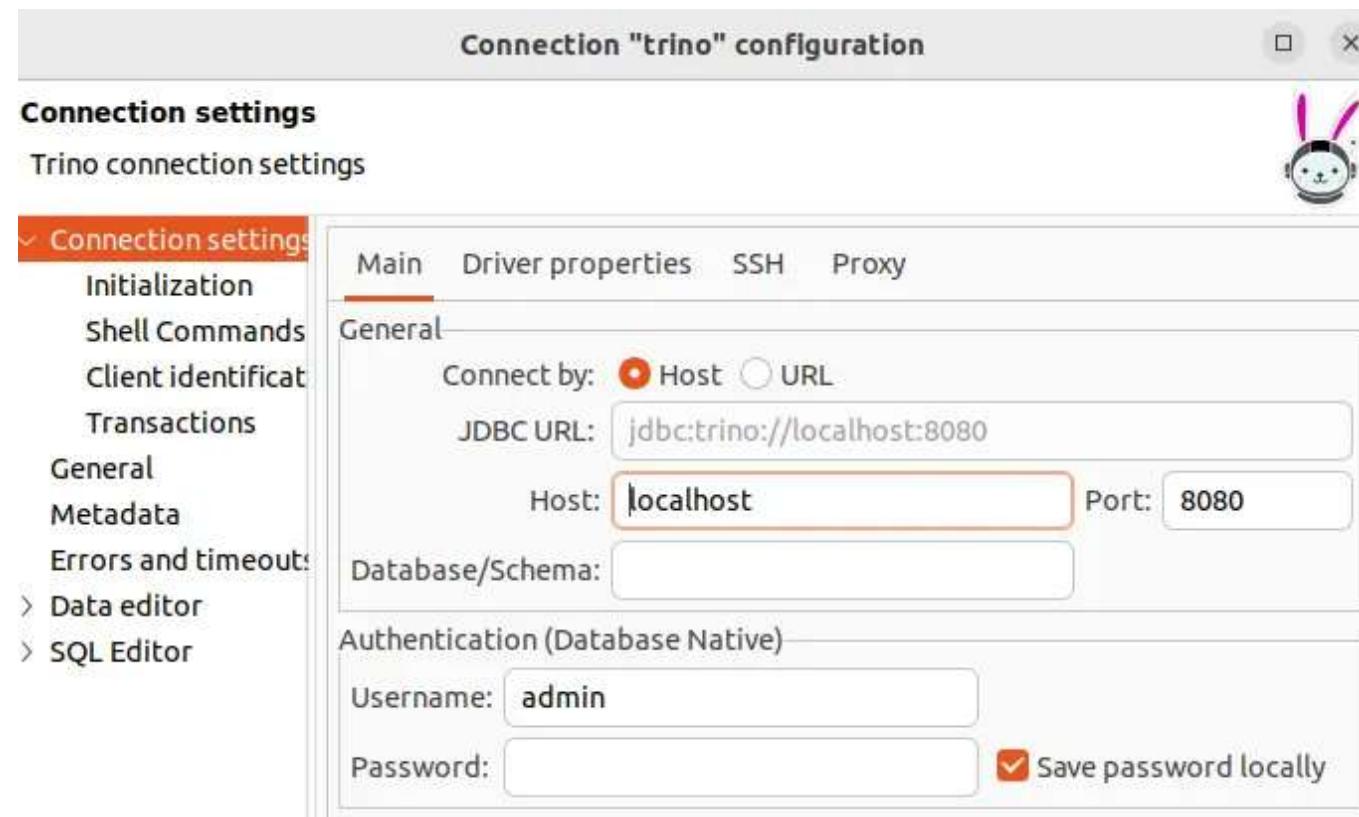
```
connector.name=mysql
connection-url=jdbc:mysql://de_mysql:3306?allowPublicKeyRetrieval=true&useSSL=false
connection-user=root
connection-password=admin
```

- PostgreSQL

```
connector.name=postgresql
connection-url=jdbc:postgresql://de_psql:5432/ecom_analytics
connection-user=admin
connection-password=admin123
```

While I would usually provide instructions for setting up the services using the [docker-compose.yml](#) file, for this particular demonstration, I'd prefer to simply explain the concept instead.

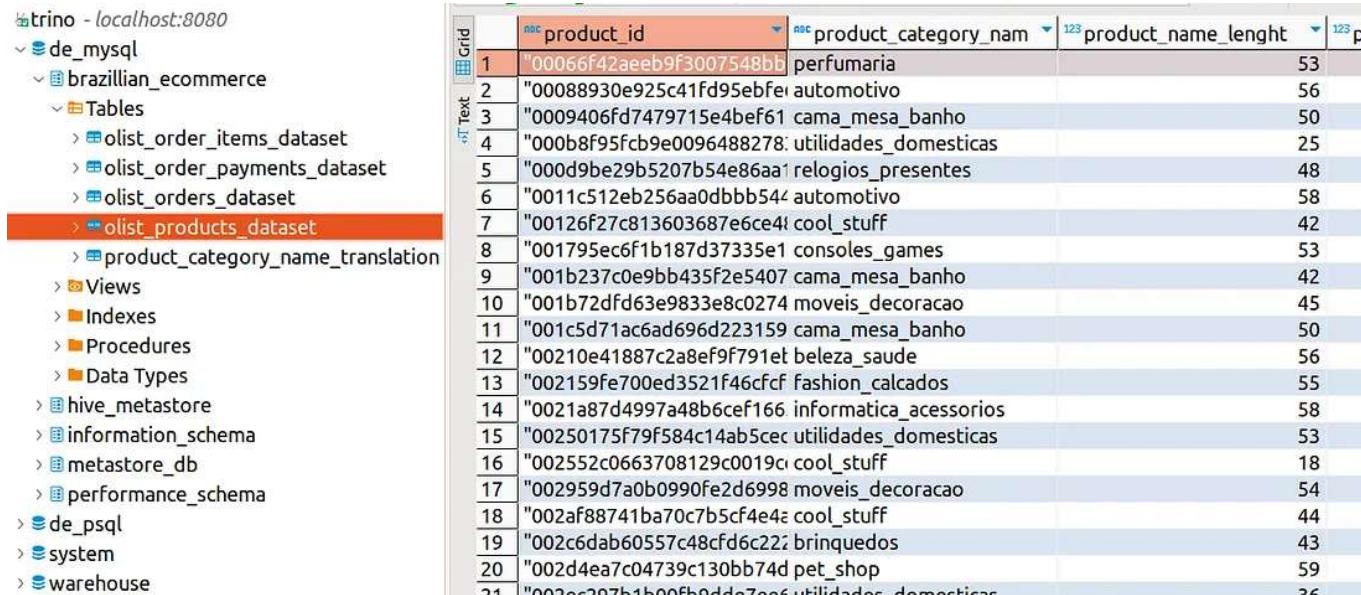
There are tools that you can use to connect to Trino such as [DBeaver](#) or [beeline CLI](#). For this demonstration, I'll be using DBeaver to establish the connection, as shown in the figure below.



Once you've established a successful connection, you should be able to view a list of available data sources as shown in the figure below.



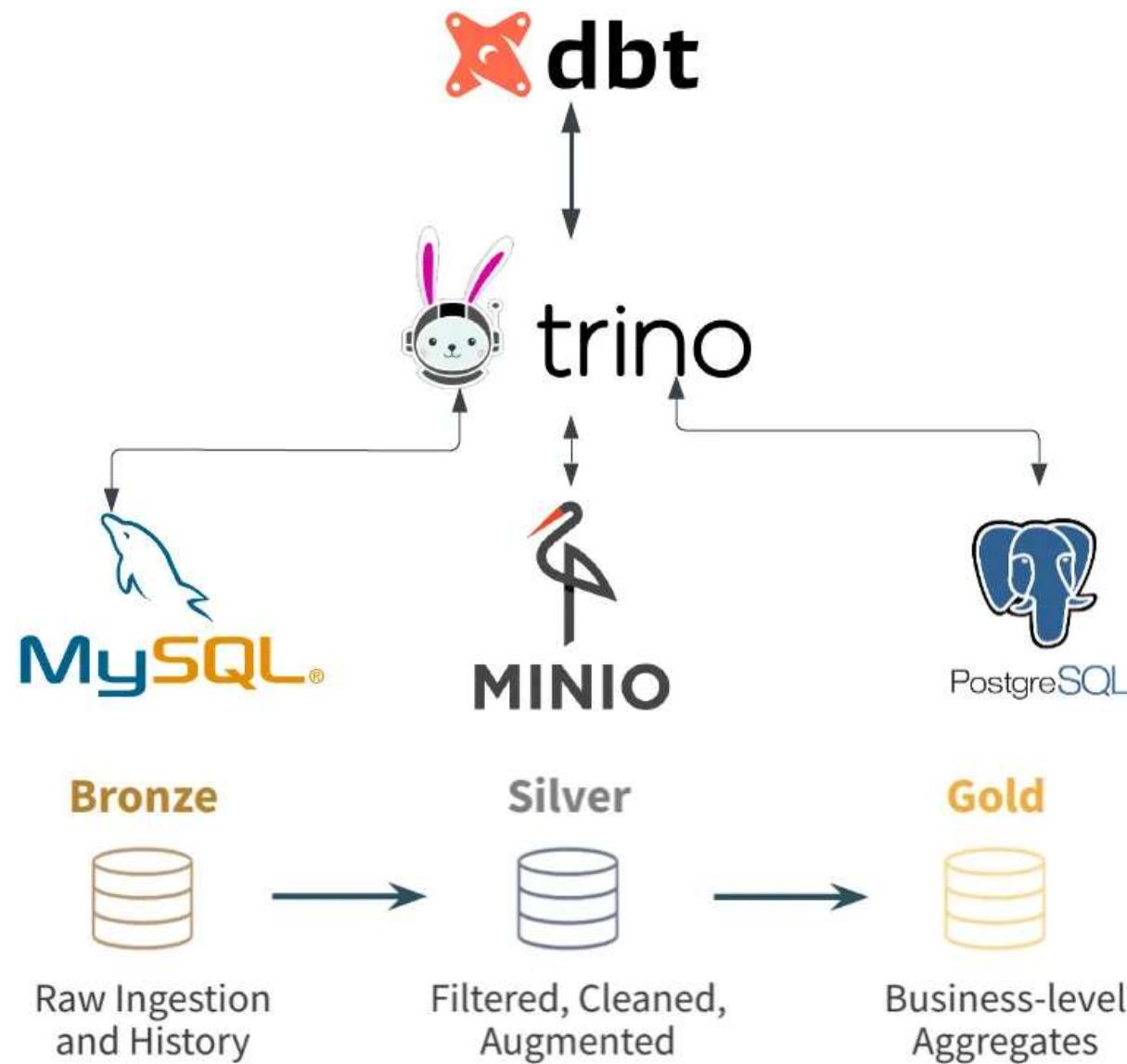
To preview the data stored in MySQL databases, you can execute a query that will display the results in a similar manner to what is shown in the figure below.



The screenshot shows a MySQL database interface with a sidebar navigation and a main data grid. The sidebar on the left lists databases, schemas, and tables. The 'olist_products_dataset' table is selected and highlighted in orange. The main grid displays the data from this table, with columns: product_id, product_category_name, product_name_length, and product_name. The data consists of 21 rows of product information.

	product_id	product_category_name	product_name_length
1	"0006f42aeeb9f3007548bb	perfumaria	53
2	"00088930e925c41fd95ebfe1	automotivo	56
3	"000940fd7479715e4bef61	cama_mesa_banho	50
4	"000b8f95fc9e0096488278	utilidades_domesticas	25
5	"000d9be29b5207b54e86aa1	relogios_presentes	48
6	"0011c512eb256aa0dbbb544	automotivo	58
7	"00126f27c813603687e6ce41	cool_stuff	42
8	"001795ec6f1b187d37335e1	consoles_games	53
9	"001b237c0e9bb435f2e5407	cama_mesa_banho	42
10	"001b72dfd63e9833e8c0274	moveis_decoracao	45
11	"001c5d71ac6ad696d223159	cama_mesa_banho	50
12	"00210e41887c2a8ef9f791et	beleza_saude	56
13	"002159fe700ed3521f46cfcf	fashion_calcados	55
14	"0021a87d4997a48b6cef166	informatica_acessorios	58
15	"00250175f79f584c14ab5cec	utilidades_domesticas	53
16	"002552c0663708129c0019ci	cool_stuff	18
17	"002959d7a0b0990fe2d6998	moveis_decoracao	54
18	"002af88741ba70c7b5cf4e4z	cool_stuff	44
19	"002c6dab60557c48cf6c22z	brinquedos	43
20	"002d4ea7c04739c130bb74d	pet_shop	59
21	"002ec297b1b00fb9ddc7ae6	utilidades_domesticas	26

DBT — the LOADER everywhere



While most people view DBT as the transformation component in an ELT pipeline, I believe that DBT can also serve as a data loader, moving data between source and target without touching any of the data definition language (DDL).

You can find a list of available [DBT adapters here](#). For the purposes of our demonstration, we will be utilizing the following adapters:

- [Starburst & Trino](#)
- [Spark](#)

By organizing your project into these files and folders, you can more easily manage your data and models, and ensure that your code is accurate and efficient.

```
ecom_analytics
  └── bronze
      ├── dbt_project.yml
      └── models
          ├── olist_order_items.sql
          ├── olist_order_payments.sql
          ├── olist_orders.sql
          └── olist_products.sql
```

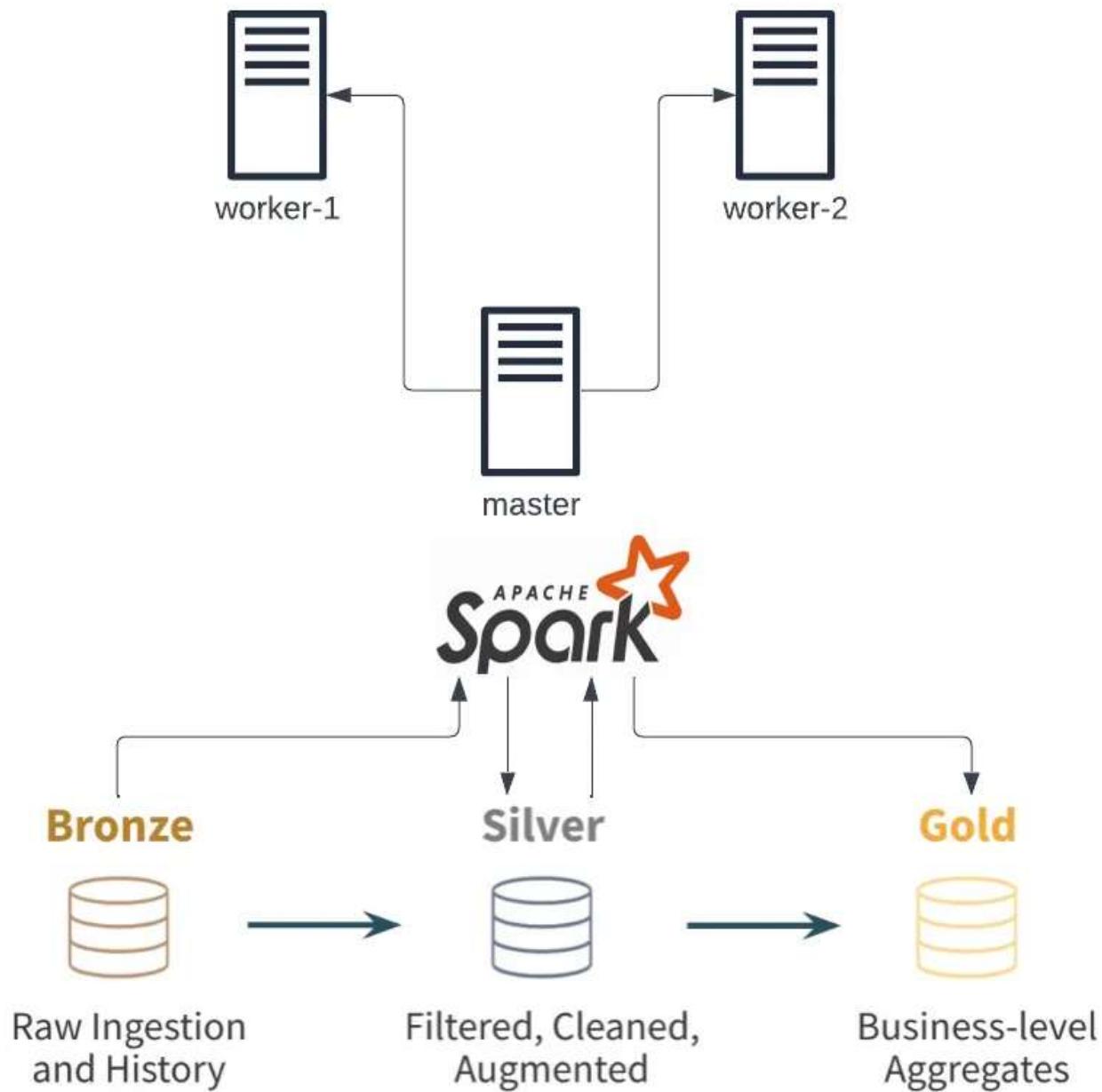
```
|- seeds
|   |- product_category_name_translation.csv
-- gold
|   |-- dbt_project.yml
|   |-- models
|       |- sales_values_by_category.sql
-- Makefile
-- profiles.yml
-- silver
|   |-- dbt_project.yml
|   |-- models
|       |- gold
|           |- sales_values_by_category.sql
|           |- orders_sources.yml
|       |- silver
|           |- dim_products.sql
|           |- fact_sales.sql
-- packages.yml
```

- `dbt_project.yml` : This is the project configuration file that defines the project name, version, and other important metadata.
- `profiles.yml` : This file contains the database connection information, such as the host, port, and credentials.
- `models/` : This folder contains the data models that you want to build, as well as the SQL code for creating those models.

- seeds/ : This folder contains CSV files in your dbt project (typically in your seeds directory), that dbt can load into your data warehouse using the `dbt seed` command.

Please refer to DBT project named [ecom_analytics](#) for more details.

Spark — the TRANSFORMER all at once



Bronze layer

We will extract the following tables from MySQL and load them to MinIO

- olist_order_items_dataset
- olist_order_payments_dataset
- olist_orders_dataset
- olist_products_dataset
- product_category_name_translation

To establish a connection to Trino for the purpose of extracting data from “catalog:de_mysql” and loading it into “catalog:warehouse”, you will need to prepare a `profiles.yml` file.

```
trino:  
  target: dev  
  outputs:  
    dev:  
      type: trino  
      user: admin  
      host: localhost  
      port: 8080  
      database: warehouse
```

```
schema: bronze
```

```
threads: 1
```

In order for DBT to know where to store the data, you will need to add these configurations to your `dbt_project.yml` file.

```
models:  
  bronze:  
    +materialized: incremental  
    +incremental_strategy: delete+insert  
    +file_format: delta  
    +location_root: s3a://warehouse/bronze
```

Go to `jdbc:trino://localhost:8080` to register `warehouse.bronze` schema.

```
CREATE SCHEMA IF NOT EXISTS warehouse.bronze  
WITH (location='s3a://warehouse/bronze');
```

Run this command to start the process

```
dbt run --project-dir ./bronze --profiles-dir ./ --full-refresh

08:00:17  Running with dbt=1.4.0
08:00:17  Found 4 models, 0 tests, 0 snapshots, 0 analyses, 310 macros, 0 operat
08:00:17
08:00:18  Concurrency: 1 threads (target='dev')
08:00:18
08:00:18  1 of 4 START sql incremental model bronze.olist_order_items .....
08:00:21  1 of 4 OK created sql incremental model bronze.olist_order_items .....
08:00:21  2 of 4 START sql incremental model bronze.olist_order_payments .....
08:00:22  2 of 4 OK created sql incremental model bronze.olist_order_payments ..
08:00:22  3 of 4 START sql incremental model bronze.olist_orders .....
08:00:23  3 of 4 OK created sql incremental model bronze.olist_orders .....
08:00:23  4 of 4 START sql incremental model bronze.olist_products .....
08:00:23  4 of 4 OK created sql incremental model bronze.olist_products .....
08:00:23
08:00:23  Finished running 4 incremental models in 0 hours 0 minutes and 6.11 se
08:00:23
08:00:23  Completed successfully
08:00:23
08:00:23  Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
```

Double check catalog:warehouse and minio:warehouse

The screenshot shows the Trino UI interface. On the left, there's a sidebar with a tree view of databases and schemas:

- trino - localhost:8080
 - > de_mysql
 - > de_sql
 - > system
 - > warehouse
 - bronze
 - Tables
 - > olist_order_items
 - > olist_order_payments
 - > olist_orders
 - > olist_products**
 - > product_category_name_translation

A preview table on the right displays 12 rows of data from the 'olist_products' table:

product_id	product_category_name	product_name_length	product_price
"00066f42aebe9f3007548bb9	perfumaria	53	53
"00088930e925c41fd95ebfe6	automotivo	56	56
"0009406fd7479715e4bef61d	cama_mesa_banho	50	50
"000b8f95fcb9e00964882783	utilidades_domesticas	25	25
"000d9be29b5207b54e86aa1	relogios_presentes	48	48
"0011c512eb256aa0dbbb544c	automotivo	58	58
"00126f27c813603687e6ce48	cool_stuff	42	42
"001795ec6f1b187d37335e1c	consoles_games	53	53
"001b237c0e9bb435f2e54071	cama_mesa_banho	42	42
"001b72dfd63e9833e8c02742	moveis_decoracao	45	45
"001c5d71ac6ad696d2231595	cama_mesa_banho	50	50
"n0210p41887r2aafqf7q1ehi	bebida_cafe	56	56

The screenshot shows the dbt UI interface for the 'warehouse' project. At the top, it says 'Created: Mon Feb 20 2023 11:40:01 GMT+0700 (Indochina Time)' and 'Access: PUBLIC 20.6 MiB - 10 Objects'.

Below that is a breadcrumb navigation bar: < warehouse / bronze. To the right are buttons for 'Rewind' (refresh), 'Refresh' (refresh), and 'Upload' (upload).

The main area shows a list of files in the 'bronze' folder:

<input type="checkbox"/>	Name	Last Modified	Size
<input type="checkbox"/>	olist_order_items		-
<input type="checkbox"/>	olist_order_payments		-
<input type="checkbox"/>	olist_orders		-
<input type="checkbox"/>	olist_products		-
<input type="checkbox"/>	product_category_name_translation		-

Silver layer

Go to `jdbc:trino://localhost:8080` to register `warehouse.silver` schema.

```
CREATE SCHEMA IF NOT EXISTS warehouse.silver
WITH (location='s3a://warehouse/silver');
```

Prepare the following packages in `packages.yml` files. These packages will help us define external tables and macros functions such as pivot commands.

```
packages:
  - package: dbt-labs/dbt_external_tables
    version: 0.8.2
  - package: dbt-labs/dbt_utils
    version: 0.9.2
```

If you are interested in understanding how dbt packages work behind the scenes, there are a few key concepts to consider.

First, a dbt package is essentially a collection of pre-built dbt models, macros, and other resources that can be used as a starting point for building out your own data pipelines. These packages can be sourced from a variety

of places, including the dbt Hub (a public repository of dbt packages) or custom-built packages created by other users.

When you install a dbt package, what you are really doing is downloading a set of YAML files that define the models, macros, and other resources that make up that package. These YAML files are then compiled into a SQL file that can be run against your database to create the relevant tables and views.

Overall, dbt packages provide a powerful way to accelerate your data pipeline development by leveraging pre-built models, macros, and other resources. By understanding how these packages work behind the scenes, you can gain a deeper appreciation for the capabilities of dbt and how it can be used to streamline your data analytics workflows.

Run this command to install dependencies

```
dbt deps --project-dir ./silver

08:14:14  Running with dbt=1.4.0
08:14:15  Installing dbt-labs/dbt_external_tables
08:14:15    Installed from version 0.8.2
08:14:15    Up to date!
08:14:15  Installing dbt-labs/dbt_utils
08:14:16    Installed from version 0.9.2
```

```
08:14:16 Updated version available: 1.0.0
08:14:16
08:14:16 Updates available for packages: ['dbt-labs/dbt_utils']
Update your versions in packages.yml, then run dbt deps
```

To connect to Apache Spark and transform data from the “bronze” layer of “catalog:warehouse” and load it back into the “silver” and “gold” layers, you will need to prepare a `profiles.yml` file.

```
spark:
  target: dev
  outputs:
    dev:
      type: spark
      method: thrift
      host: localhost
      port: 10000
      schema: silver
      connect_retries: 5
      connect_timeout: 60
      retry_all: true
```

Add these configurations to `dbt_project.yml` so that DBT has all the needed information to store the data in `catalog:warehouse`.

```
models:  
  silver:  
    +materialized: incremental  
    +incremental_strategy: merge  
    +file_format: delta  
    +pre_hook:  
      - SET spark.hadoop.fs.s3a.endpoint=http://minio:9000  
      - SET spark.hadoop.fs.s3a.access.key=minio  
      - SET spark.hadoop.fs.s3a.secret.key=minio123  
      - SET spark.hadoop.fs.s3a.path.style.access=true  
      - SET spark.hadoop.fs.s3a.connection.ssl.enabled=false  
      - SET spark.hadoop.fs.s3a.impl=org.apache.hadoop.fs.s3a.S3AFileSystem  
  silver:  
    +location_root: s3a://warehouse/silver  
  gold:  
    +location_root: s3a://warehouse/gold
```

Prepare external table in **orders_sources.yml** file

```
version: 2  
  
sources:  
  - name: silver  
    tables:  
      - name: olist_products  
        external:  
          location: 's3a://warehouse/bronze/olist_products'  
          using: delta
```

```
columns:
  - name: product_id
    data_type: string
  - name: product_category_name
    data_type: string
  - name: product_name_lenght
    data_type: int
  - name: product_description_lenght
    data_type: int
  - name: product_photos_qty
    data_type: int
  - name: product_weight_g
    data_type: int
  - name: product_length_cm
    data_type: int
  - name: product_height_cm
    data_type: int
  - name: product_width_cm
    data_type: int
- name: product_category_name_translation
  external:
    location: 's3a://warehouse/bronze/product_category_name_translation'
    using: delta
  columns:
    - name: product_category_name
      data_type: string
    - name: product_category_name_english
      data_type: string
- name: olist_order_items
  external:
    location: 's3a://warehouse/bronze/olist_order_items'
    using: delta
  columns:
    - name: order_id
      data_type: string
    - name: order_item_id
```

```
data_type: int
- name: product_id
  data_type: string
- name: seller_id
  data_type: string
- name: shipping_limit_date
  data_type: string
- name: price
  data_type: float
- name: freight_value
  data_type: float
- name: olist_order_payments
  external:
    location: 's3a://warehouse/bronze/olist_order_payments'
    using: delta
  columns:
    - name: order_id
      data_type: string
    - name: payment_sequential
      data_type: int
    - name: payment_type
      data_type: string
    - name: payment_installments
      data_type: int
    - name: payment_value
      data_type: float
- name: olist_orders
  external:
    location: 's3a://warehouse/bronze/olist_orders'
    using: delta
  columns:
    - name: order_id
      data_type: string
    - name: customer_id
      data_type: string
    - name: order_status
```

```
data_type: string
- name: order_purchase_timestamp
  data_type: string
- name: order_approved_at
  data_type: string
- name: order_delivered_carrier_date
  data_type: string
- name: order_delivered_customer_date
  data_type: string
- name: order_estimated_delivery_date
  data_type: string
```

Run this command to register the external table

```
dbt run-operation --project-dir ./silver stage_external_sources

08:15:14  Running with dbt=1.4.0
08:15:15  1 of 5 START external source silver.olist_products
08:15:15  1 of 5 (1) refresh table silver.olist_products
08:15:15  1 of 5 (1) OK
08:15:15  2 of 5 START external source silver.product_category_name_translation
08:15:15  2 of 5 (1) refresh table silver.product_category_name_translation
08:15:15  2 of 5 (1) OK
08:15:15  3 of 5 START external source silver.olist_order_items
08:15:16  3 of 5 (1) refresh table silver.olist_order_items
08:15:16  3 of 5 (1) OK
08:15:16  4 of 5 START external source silver.olist_order_payments
08:15:16  4 of 5 (1) refresh table silver.olist_order_payments
08:15:16  4 of 5 (1) OK
```

```
08:15:16 5 of 5 START external source silver.olist_orders
08:15:16 5 of 5 (1) refresh table silver.olist_orders
08:15:17 5 of 5 (1) OK
```

The command `docker logs spark-thrift-server` can be used to view the logs of the Spark Thrift Server container. This will show what the Spark Thrift Server is doing behind the scenes, including any error messages, warnings, or informational messages. The logs can be used to troubleshoot issues or gain insight into the behavior of the Spark Thrift Server. For example, this is how `dbt-labs/dbt_external_tables` execute.

```
create table silver.olist_order_items (
    order_id string,
    order_item_id int,
    product_id string,
    seller_id string,
    shipping_limit_date string,
    price float,
    freight_value float
) using delta
location 's3a://warehouse/bronze/olist_order_items'
```

Run this command to start the process

```
dbt run --project-dir ./silver --profiles-dir ./ --full-refresh

04:32:52  Running with dbt=1.4.0
04:32:52  Found 3 models, 0 tests, 0 snapshots, 0 analyses, 570 macros, 0 operat
04:32:52
04:32:52  Concurrency: 1 threads (target='dev')
04:32:52
04:32:52  1 of 3 START sql incremental model silver.dim_products .....
04:32:59  1 of 3 OK created sql incremental model silver.dim_products .....
04:32:59  2 of 3 START sql incremental model silver.fact_sales .....
04:33:08  2 of 3 OK created sql incremental model silver.fact_sales .....
04:33:08  3 of 3 START sql incremental model silver.sales_values_by_category ...
04:33:13  3 of 3 OK created sql incremental model silver.sales_values_by_category
04:33:13
04:33:13  Finished running 3 incremental models in 0 hours 0 minutes and 21.73 s
04:33:13
04:33:13  Completed successfully
04:33:13
04:33:13  Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
```

Double check **catalog:warehouse** and **minio:warehouse**

The screenshot shows the Trino interface with a sidebar on the left displaying the schema tree:

- trino - localhost:8080
- > de_mysql
- > de_postgres
- > system
- > warehouse
 - > bronze
 - > default
 - > information_schema
 - > silver
 - > Tables
 - > abc_by_category
 - > dim_products
 - > fact_sales
 - > monthly_sales_category
 - > olist_order_payments
 - > olist_orders
 - > olist_products
 - > product_category_name_translation
 - > sales_values_by_category**
 - > sales_values_overview

To the right of the schema tree is a data grid titled "Grid" with the following columns: daily, monthly, category, sales, and bills. The "daily" column is currently selected. The data grid contains 21 rows of sales data:

	daily	monthly	category	sales	bills
1	2017-07-29	2017-07	watches_gifts	868.19	1
2	2017-06-12	2017-06	housewares	63.44	1
3	2018-06-15	2018-06	furniture_decor	44.99	1
4	2018-08-22	2018-08	food	81.86	1
5	2017-04-03	2017-04	garden_tools	100.37	2
6	2017-12-14	2017-12	computers_accessories	46.69	1
7	2018-08-21	2018-08	sports_leisure	138.28	1
8	2017-10-27	2017-10	bed_bath_table	117.85	1
9	2018-07-23	2018-07	auto	194.64	2
10	2017-11-24	2017-11	health_beauty	649.61	6
11	2018-05-05	2018-05	health_beauty	511.75	1
12	2018-05-09	2018-05	telephony	37.38	1
13	2018-01-30	2018-01	art	62.78	1
14	2017-05-29	2017-05	electronics	112.81	1
15	2017-12-05	2017-12	toys	517.44	1
16	2017-11-24	2017-11	computers_accessories	398.8	1
17	2017-12-16	2017-12	computers_accessories	69.24	1
18	2018-06-24	2018-06	home_appliances	48.21	1
19	2018-07-26	2018-07	signaling_and_security	258.56	1
20	2017-12-08	2017-12	toys	62.77	1
21	2017-02-13	2017-02	furniture_decor	45.95	1

The screenshot shows the Minio interface with a path bar: warehouse / silver. The interface includes buttons for Rewind, Refresh, and Upload.

The file listing shows the following files:

Name	Last Modified	Size
dim_products		-
fact_sales		-

Gold layer

Great, it's good to know that you were able to load the gold data layer from minio:warehouse into PostgreSQL for analytics. By running this command,

you finish the pipeline

```
dbt run --project-dir ./gold --profiles-dir ./ --full-refresh

03:49:00  Running with dbt=1.4.0
03:49:00  Found 1 model, 0 tests, 0 snapshots, 0 analyses, 310 macros, 0 operations
03:49:00
03:49:01  Concurrency: 1 threads (target='dev')
03:49:01
03:49:01  1 of 1 START sql table model gold.sales_values_by_category .....
03:49:01  1 of 1 OK created sql table model gold.sales_values_by_category .....
03:49:01
03:49:01  Finished running 1 table model in 0 hours 0 minutes and 0.82 seconds (0.82s)
03:49:01
03:49:01  Completed successfully
03:49:01
03:49:01  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
```

Double check catalog:de_psql and minio:warehouse

The screenshot shows the Trino UI interface. On the left, there is a sidebar with a tree view of databases and schemas:

- trino - localhost:8080
- > de_mysql
- > de_psql
- > gold
- Tables
 - abc_by_category
 - monthly_sales_category
 - sales_values_by_category
 - sales_values_overview

The 'sales_values_by_category' table is highlighted with a red rectangle. To the right of the sidebar is a detailed view of the table's data in a grid format:

	daily	monthly	category	sales	bills	values_per
1	2017-07-29	2017-07	watches_gifts	868.19	1	
2	2017-06-12	2017-06	housewares	63.44	1	
3	2018-06-15	2018-06	furniture_decor	44.99	1	
4	2018-08-22	2018-08	Food	81.86	1	
5	2017-04-03	2017-04	garden_tools	100.37	2	
6	2017-12-14	2017-12	computers_accessories	46.69	1	
7	2018-08-21	2018-08	sports_leisure	138.28	1	
8	2018-06-27	2018-06	furniture_decor	233.84	1	

The screenshot shows the Databricks UI interface. At the top, there's a header with the word "warehouse". Below it, a message says "Created: Mon Feb 20 2023 11:40:01 GMT+0700 (Indochina Time) Access: PUBLIC 51.4 MiB - 30 Objects". To the right are three buttons: "Rewind" with a circular arrow icon, "Refresh" with a circular arrow icon, and "Upload" with an upward arrow icon. Below the header is a breadcrumb navigation bar with a back arrow, the path "warehouse / gold", and a "Create new path" button. The main area is a table with columns: "Name" (with a checkbox and a triangle icon), "Last Modified", and "Size". There is one entry: "sales_values_by_category" with a size of "-".

Upon inspecting the Spark jobs by navigating to `localhost:4040/jobs`, it becomes evident that DBT has effectively utilized the power of computing from our Spark mini-cluster.

The screenshot shows the Apache Spark 3.3.1 UI. At the top, there's a navigation bar with tabs: "Jobs" (which is selected), "Stages", "Storage", "Environment", "Executors", "SQL / DataFrame", and "JDBC/ODBC Server". To the right of the tabs is a link "Thrift JDBC/ODBC Server application UI". Below the navigation bar is a section titled "Spark Jobs (?)". It displays the following information:

- User: root
- Total Uptime: 3.2 min
- Scheduling Mode: FIFO
- Completed Jobs: 53

Below this, there are two sections: "Event Timeline" (which is collapsed) and "Completed Jobs (53)" (which is expanded). The "Completed Jobs" section includes a page navigation bar with "Page: 1" and a "Go" button, and a table with the following columns: "Job Id (Job Group)", "Description", "Submitted", "Duration", "Stages: Succeeded/Total", and "Tasks (for all stages): Succeeded/Total". The table contains five rows of data, each representing a completed job with its ID, description (including code snippets), submission time, duration, stages completed, and total tasks completed.

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
52 (ed71ccb9-b424-4b4d-b71b-6339bace6096)	Delta: /* ("app": "dbt", "dbt_version": "1.4.0", "profile_name": "spark", "target_name": "dev", "n... toString at String.java:2994	2023/02/22 04:53:55	36 ms	1/1 (2 skipped)	1/1 (52 skipped)
51 (ed71ccb9-b424-4b4d-b71b-6339bace6096)	Delta: /* ("app": "dbt", "dbt_version": "1.4.0", "profile_name": "spark", "target_name": "dev", "n... toString at String.java:2994	2023/02/22 04:53:54	0.8 s	1/1 (1 skipped)	50/50 (2 skipped)
50 (ed71ccb9-b424-4b4d-b71b-6339bace6096)	Delta: /* ("app": "dbt", "dbt_version": "1.4.0", "profile_name": "spark", "target_name": "dev", "n... toString at String.java:2994	2023/02/22 04:53:54	71 ms	1/1	2/2
49 (ed71ccb9-b424-4b4d-b71b-6339bace6096)	/* ("app": "dbt", "dbt_version": "1.4.0", "profile_name": "spark", "target_name": "dev", "node_i... \$anonfun\$recordDeltaOperationInternal\$1 at DatabricksLogging.scala:128	2023/02/22 04:53:53	0.3 s	1/1 (1 skipped)	50/50 (2 skipped)
48 (ed71ccb9-b424-4b4d-b71b-6339bace6096)	/* ("app": "dbt", "dbt_version": "1.4.0", "profile_name": "spark", "target_name": "dev", "node_i... \$anonfun\$recordDeltaOperationInternal\$1 at DatabricksLogging.scala:128	2023/02/22 04:53:53	0.5 s	1/1 (3 skipped)	1/1 (6 skipped)

Executors

Show 20 entries Search:

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(3)	750	36.9 MiB / 1.1 GiB	0.0 B	2	0	0	1094	1094	6.4 min (3 s)	22.9 MiB	11.2 MiB	11.2 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(3)	750	36.9 MiB / 1.1 GiB	0.0 B	2	0	0	1094	1094	6.4 min (3 s)	22.9 MiB	11.2 MiB	11.2 MiB	0

Executors

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
0	172.26.0.9:46081	Active	309	12.1 MiB / 366.3 MiB	0.0 B	1	0	0	462	462	24 s (0.7 s)	16.6 MiB	41 KiB	37.6 KiB	stdout stderr	Thread Dump
driver	dcaaf327064a:41833	Active	0	12.6 MiB / 366.3 MiB	0.0 B	0	0	0	0	0	5.5 min (1 s)	0.0 B	0.0 B	0.0 B		Thread Dump
1	172.26.0.10:34293	Active	441	12.1 MiB / 366.3 MiB	0.0 B	1	0	0	632	632	27 s (0.8 s)	6.2 MiB	11.1 MiB	11.1 MiB	stdout stderr	Thread Dump

Conclusion

In summary, the synergy of Trino, DBT, and Spark presents a formidable solution for data extraction, transformation, and loading (ETL), with a particular focus on the ELT process. Collectively, these three components constitute an excellent choice for analytics engineers seeking a seamless and dependable approach to handle and analyze their data.

Trino simplifies data connectivity across diverse sources, while DBT's versatility shines in facilitating data loading and transfer between disparate systems. Spark, with its impressive ability to transform substantial datasets,

seamlessly integrates into this trio of tools. Together, they harmoniously complement each other, culminating in an efficient and adaptable data pipeline.

P/S: The title of the article “Everything, Everywhere, All at Once” is borrowed from a movie.



References

- [Federated queries in data lakes with Redpanda and Trino](#)
- [Kaggle: Brazilian E-Commerce Public Dataset by Olist](#)
- [Iceberg + Spark + Trino + Dagster: modern, open-source data stack demo](#)

Big Data Analytics

Data Migration

Federated Queries

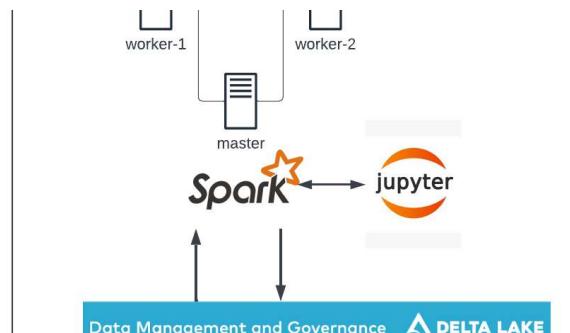


Written by Ong Xuan Hong

136 Followers

Follow

More from Ong Xuan Hong



Ong Xuan Hong

DataOps 02: Spawn up Apache Spark infrastructure by using...

When working on real data products, we will register an account on cloud providers such...

8 min read · Jan 28, 2023



30



1



Dagster - DataOps and MLOps for Machine Learning Engineers

Ong Xuan Hong

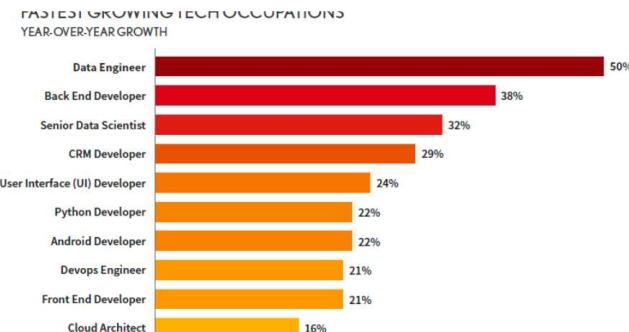
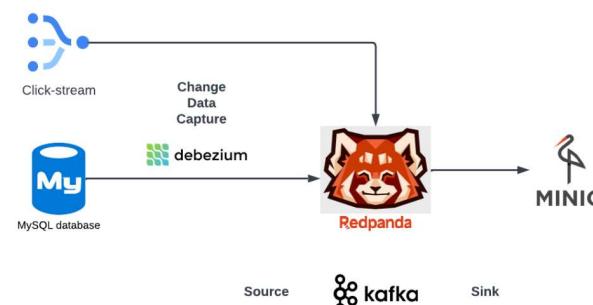
MLOps 02: Dagster—Seamless Data Pipelines & ML Model...

Thanks to everyone who joined my sharing session today on Data Ops & MLOps for...

4 min read · Aug 14, 2023



1





Ong Xuan Hong

DataOps 01: Stream data ingestion with Redpanda

In the article What's Next for Data Engineering in 2023? 7 Predictions , one of...

10 min read • Feb 11, 2023



10



Ong Xuan Hong

Why should you start a career in the Data Science field with a Data...

Greetings, dear readers! It's been quite a while since I last penned down my thoughts ...

11 min read • Feb 1, 2023

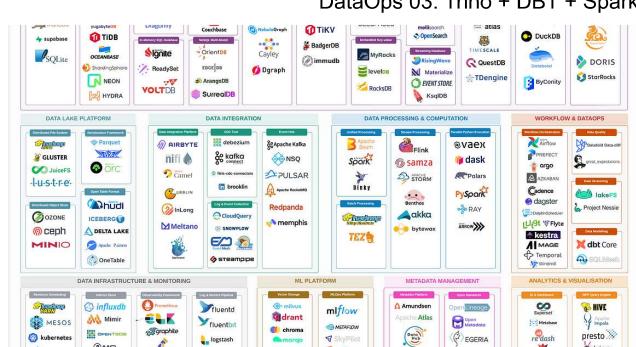


10



See all from Ong Xuan Hong

Recommended from Medium



Alireza Sadeghi

Open Source Data Engineering Landscape 2024

Exploration of the open source software in data engineering ecosystem

11 min read · Feb 4, 2024



328



9



9 min read · Feb 5, 2024



929



12



Lists



Staff Picks

584 stories · 760 saves



Stories to Help You Level-Up at Work

19 stories · 485 saves



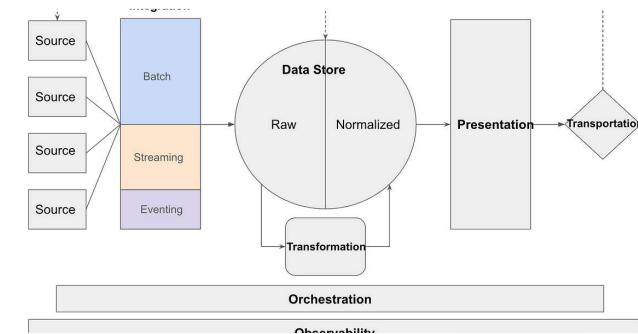
Self-Improvement 101

20 stories · 1359 saves



Productivity 101

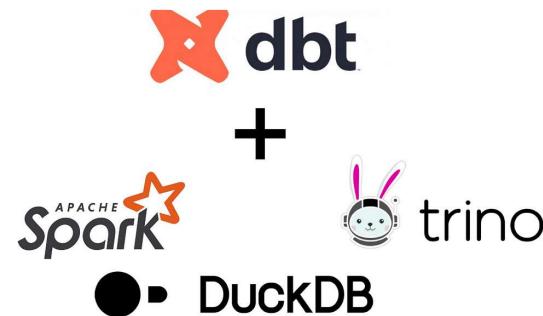
20 stories · 1247 saves



Dave Melillo in Towards Data Science

Building a Data Platform in 2024

How to build a modern, scalable data platform to power your analytics and data...



 Niels Claeys in datamindedbe

Head-to-head comparison of dbt SQL engines

Compare usage and performance of dbt against 3 popular open-source SQL engines...

8 min read · Sep 8, 2023

👏 467

💬 5



DATA LAKEHOUSE BYTES

with Alex Merced

Partitioning Practices in APACHE HIVE AND APACHE ICEBERG



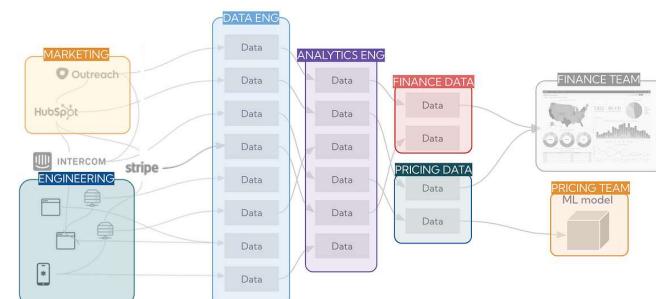
 Alex Merced - Tech Evang... in Data Lakehouse B...

Partitioning Practices in Apache Hive and Apache Iceberg

Introduction

6 min read · 6 days ago

👏 1





Thibault Latrace in Back Market Blog



Mikkel Dengsøe

Back Market's journey towards data self-service

Or the importance of internal data training

7 min read · Feb 8, 2024



10



3



141



11 min read · Feb 12, 2024

See more recommendations

[Help](#) [Status](#) [About](#) [Careers](#) [Blog](#) [Privacy](#) [Terms](#) [Text to speech](#) [Teams](#)