

# Build a simple Data Lakehouse with Fivetran and Databricks

[#databricks](#) [#fivetran](#) [#dbt](#) [#mds](#)

The Modern Data Stack (MDS) is gaining increasing popularity in recent years along with cloud computing. MDS put the cloud data warehouse such as Snowflake and Databricks at its core and uses modern data integration tools to load data into the cloud data warehouse.

In the post, I will demonstrate how to use some emerging MDS tools on the market to build a simple data lakehouse.

## The tools I use are:

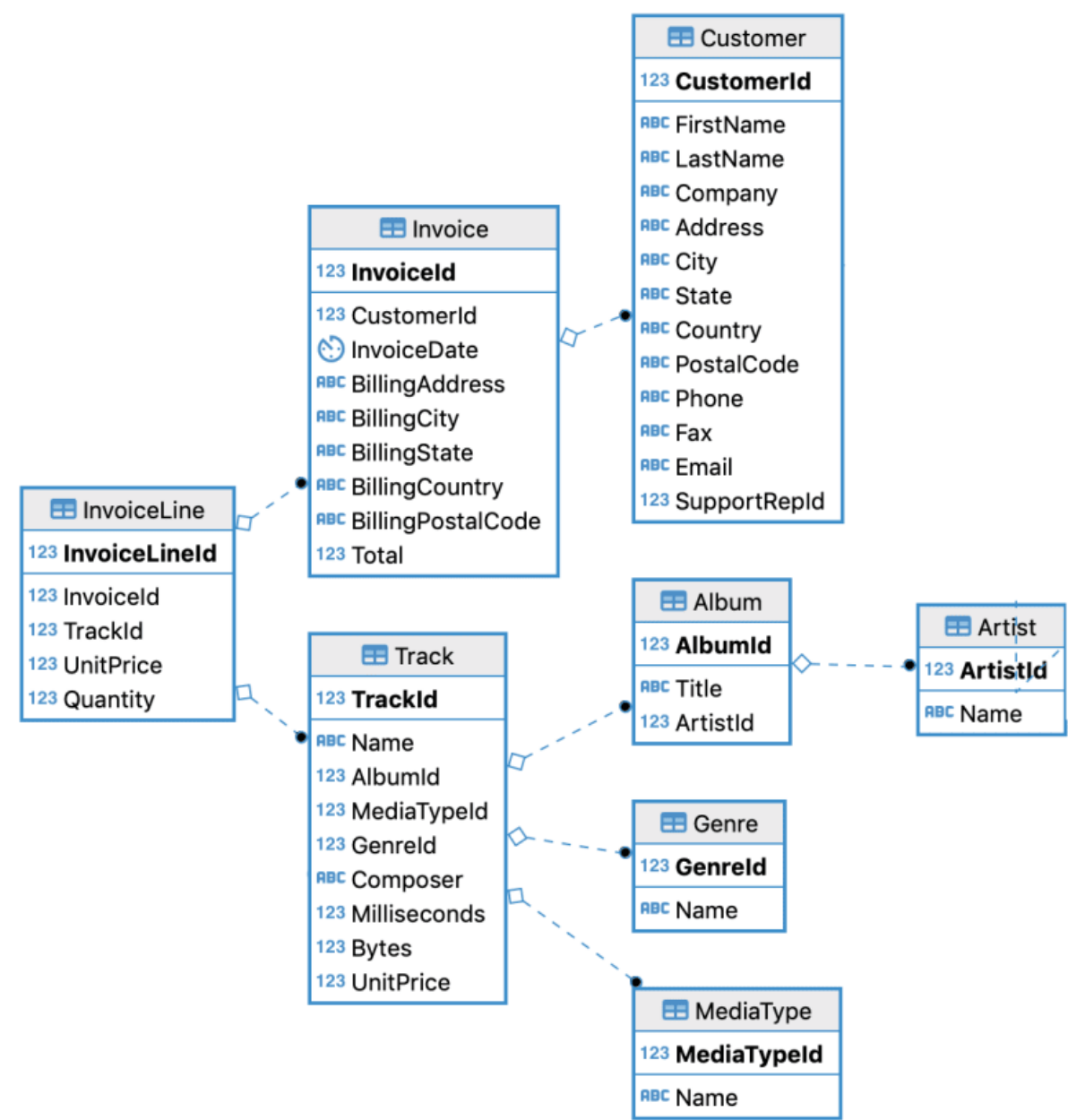
- Fivetran - a leading cloud-based data integration tool. It offers many connectors to help connect to the data source and load the data into cloud data storage.
- Databricks - Databricks is a well-known product in the AI/ML space for many years, and they launched Delta Lake - an open-source application in 2019. You can build a Data Lakehouse with it.
- Postgres - A open-source transactional database. I will use it as my source data database. The source data I use for this demo is called Chinook database. You can find the data [here](#).
- I have launched an AWS RDS instance to host the database to let Fivetran read the data.
- dbt - an open-source application to build the data models.

The Tool

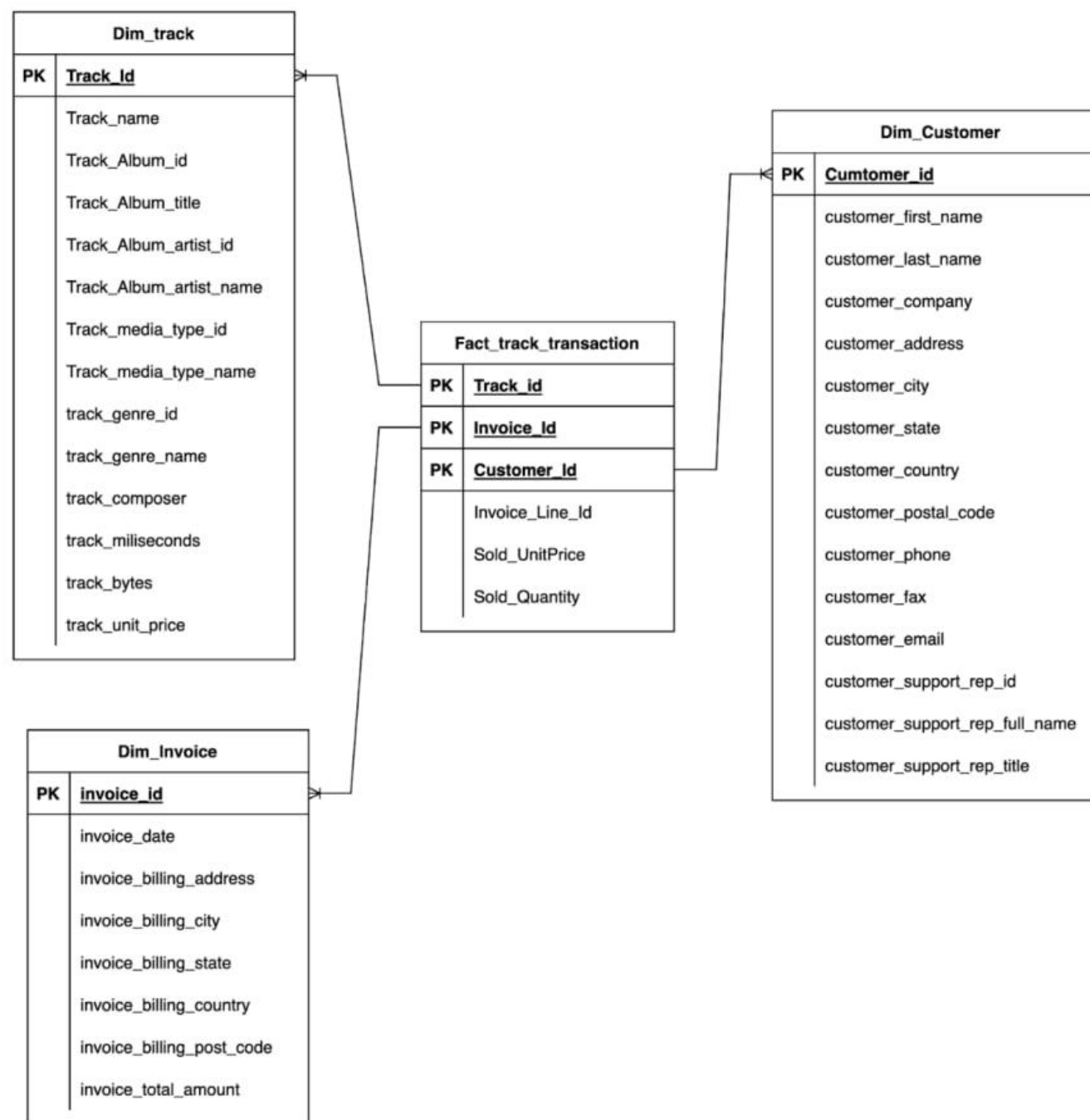
# The task

In this demo, I will convert the OLTP database into a star schema.

## The Source schema:



## The target schema



## Lakehouse layers design

The lakehouse contains the following layers:

### Landing Layer

The landing layer contains all the data loaded by Fivetran. Tables have additional metadata fields for tracking purposes. All the data should be in their original format.

### History Layer

The history layer contains history tables using SCD2 methodology. Every table in the landing area has a Historical table

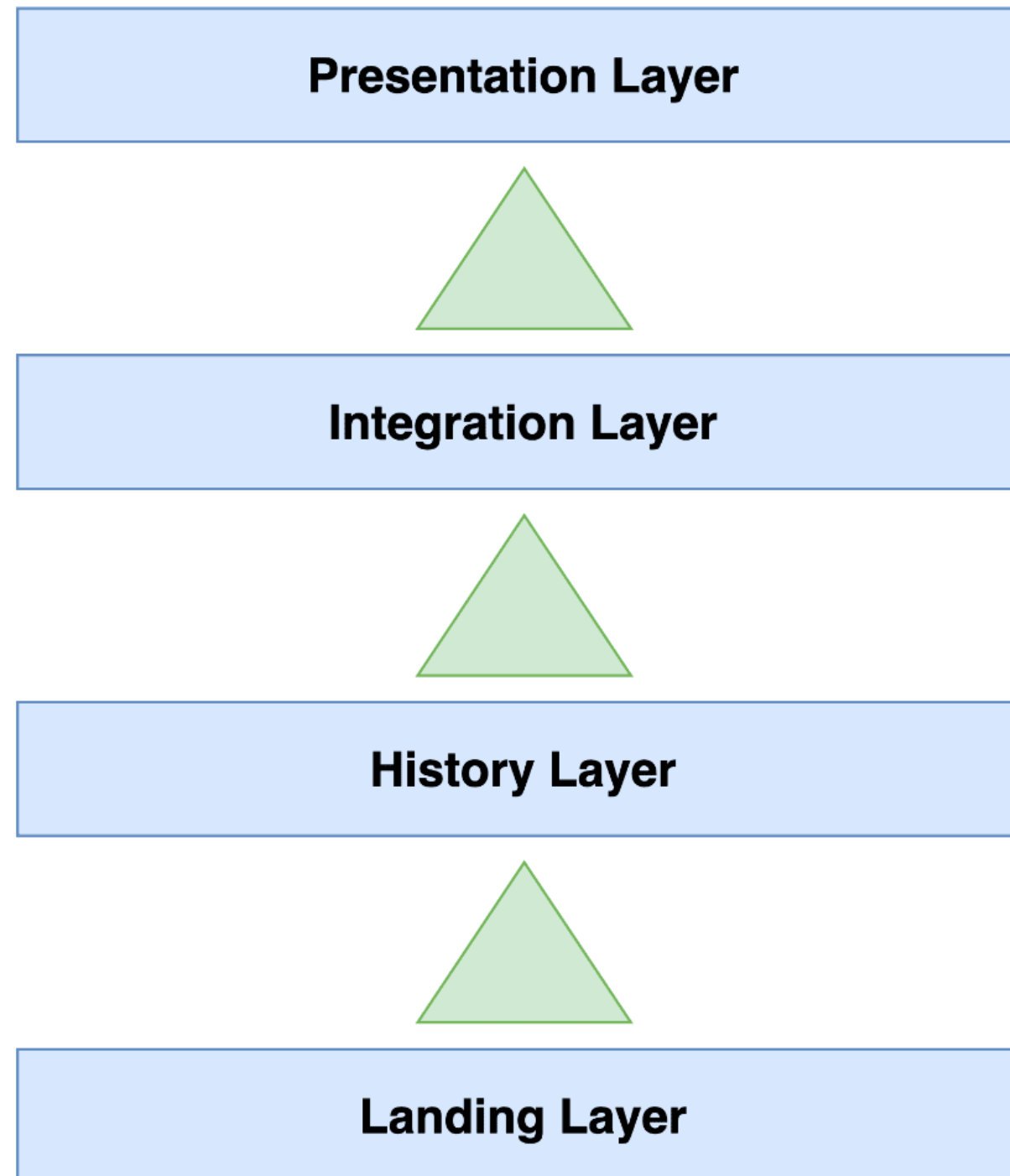
table in the landing area has a historical table.

### **Integration Layer**

The integration layer contains temporary tables to associate transformation. This is where you denormalize the source data and apply business rules.

### **Presentation Layer**

The presentation layer keeps the transformed and business-friendly star schemas, in other words: the data warehouse.



## **The build**

**Step 1: Set up the environment**

Step 1: Set up the environment

AWS RDS

As I mentioned before, I have used the AWS RDS to host a Postgres database and create the Chinook database. Allow For Public Access option is selected.

Databricks

First, create a cluster on your Databricks account (I am using Databricks on Azure). There are two types of clusters. For this practice, I created a single node cluster.

Configuration   Notebooks (0)   Libraries   Event log   Spa

Cluster mode ?

Single Node

Databricks Runtime Version

10.4 LTS (includes Apache Spark 3.2.1, Scala 2.12)

Step 2: use Fivetran to load the data into Databricks

Set up the destinations

Server Host Name:	adb-[REDACTED]azuredatabricks.net
Port:	443
Http Path:	sql/protocolv1/o/[REDACTED]25-
Personal Access Token:	*****
Create External Tables:	false
External Location:	
Default Time Zone:	Australian Eastern Standard Time AEST (UTC +10)
Data processing location:	Australia
Cloud service provider:	Azure
Cloud region:	australiaeast
Destination Group ID:	granules_basket

The Port and Http path can be found under Databricks cluster attribute.

Set up the connector

1. Select Postgres RDS instance

1. Select Postgres RDS instance

2. Set the schema prefix. I named it as source\_fivetran\_pg. This indicates the data in this schema are extracted from Fivetran and in their original format

### Test the connection

During the connection setup, Fivetran may prompt you to select the certification, select the root level certification.

### Strat Initial sync

Fivetran will start the initial sync after it can connect to the database. Select the table or schema you want to sync, the leave the rest to Fivetran.

▼ public

☒ 11 tables

☒ ▶ Album

☒ ▶ Artist

☒ ▶ Customer

☒ ▶ Employee

☒ ▶ Genre

☒ ▶ Invoice

☒ ▶ InvoiceLine

☒ ▶ MediaType

☒ ▶ Playlist

☒ ▶ PlaylistTrack

☒ ▶ Track

Check your data on Databricks

Fivetran will create a new schema in Databricks called source\_fivetran\_pg\_chinook\_public, and all the tables I selected are in this schema.

Tables		
🔍 Filter Tables		
📊 album		▼
📊 artist		▼
📊 customer		▼
📊 employee		▼
📊 fivetran_audit		▼
📊 genre		▼
📊 invoice		▼
📊 invoiceline		▼
📊 mediatype		▼
📊 playlist		▼
📊 playlisttrack		▼
📊 track		▼


Step 3: build the history tables

Fivetran creates two metadata columns in each source table: \_fivetran\_deleted and \_fivetran\_synced. \_fivetran\_synced field contains the timestamp when Fivetran load the data


DataEngBytes

Trending on DEV Community👤


👤🔥



What was your win this week?  
#discuss #weeklyretro



Meme Monday🐞  
#watercooler #discuss #jokes



I updated my Portfolio (based on Feedback)  
#webdev #showdev #career #javascript

into Databricks, I will use the column and the primary key of each table to create the history table.

In dbt, create the snapshot model file and use the config function to define the schema name of the history table, unique\_key, update\_at and file\_format. The file\_format parameter must be set to 'delta', dbt will create a delta table in Databricks.

Example of snapshot model in dbt:

```
{% snapshot CHINOOK_GENRE %}

{{
  config(
    target_schema='dev_dlh_hist_chinook',
    unique_key='genreid',
    strategy='timestamp',
    updated_at='_FIVETRAN_SYNCED',
    file_format='delta',
  )
}}

select * from {{ source('chinook_landing', 'genre') }}

{% endsnapshot %}
```

Then run the 'dbt snapshot' command. It will create the history table in the target schema.

	<div><div></div>chinook_album</div>
	<div><div></div>chinook_artist</div>
	<div><div></div>chinook_customer</div>
	<div><div></div>chinook_employee</div>
	<div><div></div>chinook_genre</div>
	<div><div></div>chinook_invoice</div>
	<div><div></div>chinook_invoiceline</div>
	<div><div></div>chinook_mediatype</div>
	<div><div></div>chinook_playlist</div>
	<div><div></div>chinook_playlisttrack</div>
<div><div></div>dev_dlh_hist_chinook</div>	



This is an example of the history Artist table (dbt creates 4 additional metadata columns to maintain the history of each table)

Schema:

col_name	data_type	comment
name	string	
_fivetran_deleted	boolean	
_fivetran_synced	timestamp	
dbt_scd_id	string	
dbt_updated_at	timestamp	
dbt_valid_from	timestamp	
dbt_valid_to	timestamp	

Showing all 11 rows.

Sample Data:

artistid	name	_fivetran_deleted	_fivetran_synced	dbt_scd_id	dbt_updated_at
155	Zeca Pagodinho	null	2022-05-19T11:10:14.085+0000	572a7b43b6cae8ba0dbe0dfca20836a4	2022-05-19T11:10:14.085+0000
168	Yousou N'Dour	null	2022-05-19T11:10:14.088+0000	22727831d43c9588b3e30d44790eccf2	2022-05-19T11:10:14.088+0000
212	Yo-Yo Ma	null	2022-05-19T11:10:14.097+0000	2fd80594a52410e0887e2959e13b840	2022-05-19T11:10:14.097+0000
255	Yehudi Menuhin	null	2022-05-19T11:10:14.106+0000	583c0689b6d66c721370122bd12f2be2b	2022-05-19T11:10:14.106+0000
181	Xis	null	2022-05-19T11:10:14.091+0000	5a8f3e2d5dbd9c6526120d3b65dd5570	2022-05-19T11:10:14.091+0000
211	Wilhelm Kempff	null	2022-05-19T11:10:14.097+0000	a518e81383aa71d9451c1f19fbaa9e1a	2022-05-19T11:10:14.097+0000
154	Whitesnake	null	2022-05-19T11:10:14.084+0000	aa734cfe773be70c30f28ad1e13fe850	2022-05-19T11:10:14.084+0000

Showing all 275 rows.

#### Step 4: Integrate the data

After the history tables have been created, we can start to join the tables together, apply the business logic and create the dimension table and fact tables.

The objects I created in this schema are views, not tables. They only contain the current values of each dimension and fact table.

Search	Filter Tables
dev_dlh_stg	vm_curr_dim_customer
dev_dlh_stg_chinook	vm_curr_dim_invoice
	vm_curr_dim_track
	vm_curr_fact_track_transaction

#### Step 5: create the star schema

Finally, we can create the star schema. Once again, I use the incremental functionality from dbt to build the table. Each dimension table has start\_timestamp and end\_timestamp to maintain the history.

dev\_dlh\_star

dev\_dlh\_star\_schema

Filter Tables

agg\_tran

dim\_customer

dim\_invoice

dim\_track

fact\_track\_transaction

Example of the dim\_track table:

Schema:

	col_name	data_type	comment
1	track_id	int	
2	track_name	string	
3	track_album_id	int	
4	track_album_title	string	
5	track_album_artist_id	int	
6	track_album_artist_name	string	
7	track_media_type_id	int	

Showing all 17 rows.

Sample Data:

	track_id	track_name	track_album_id	track_album_title	track_album_artist_id	track_album_artist_name
1	1	For Those About To Rock (We Salute You)	1	For Those About To Rock We Salute You	1	AC/DC
2	2	Balls to the Wall	2	Balls to the Wall	2	Accept
3	3	Fast As a Shark	3	Restless and Wild	2	Accept
4	4	Restless and Wild	3	Restless and Wild	2	Accept
5	5	Princess of the Dawn	3	Restless and Wild	2	Accept
6	6	Put The Finger On You	1	For Those About To Rock We Salute You	1	AC/DC
7	7	Let's Get It Up	1	For Those About To Rock We Salute You	1	AC/DC

Truncated results, showing first 1000 rows.

## Conclusion

MDS makes building the data lakehouse and ELT a lot easier. Fivetran is great for loading the data from the source systems into the cloud data warehouse; Databricks is very powerful for data insertion, selection, and table join.

Top comments (0)

[Code of Conduct](#) • [Report abuse](#)

mode