

# What is SPARK SQL?

Spark SQL is a module in Apache Spark that lets you run SQL queries on your big data.

It's like giving Spark the power of SQL!

You can write SQL queries directly – just like you do in a database – but the magic is:

It runs on top of Spark's distributed engine. So, it's fast and scalable

## Why SPARK SQL?

- ✓ SQL is easy to learn – even if you're not a programmer.
- ✓ It's great for ad-hoc analysis – just like querying a database.
- ✓ It's widely used by analysts, data scientists, and engineers – all in one environment!

### Temp Views

```
▶ ✓ May 04, 2025 (1s)

%python
df.createOrReplaceTempView("orders_temp")

▶ ✓ May 04, 2025 (6s)

SELECT * FROM orders_temp;
```

### Global Temp Views

```
▼ ✓ May 04, 2025 (<1s)

%python

df.createOrReplaceGlobalTempView("orders_global_temp")
```

Temp views are only available for the Notebook session, Global views remain till the cluster is not killed, available in all the sessions of the same cluster.

## Managed Tables

▶ ▾ ✓ May 04, 2025 (7s)

```
CREATE TABLE sparksql_cata.sparksql_schema.order_man
AS
SELECT * FROM orders_temp;
```

▶ 📄 \_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num\_affected\_row

Table ▾ +

1 <sup>2</sup> <sub>3</sub> num_affected_rows	1 <sup>2</sup> <sub>3</sub> num_inserted_rows
---	---

▶ ✓ May 04, 2025 (2s)

```
DROP TABLE sparksql_cata.sparksql_schema.order_man;
```

▶ ✓ May 04, 2025 (2s)

```
UNDROP TABLE sparksql_cata.sparksql_schema.order_man;
```

UNDROP is a unique functionality in Managed Tables, even after deletion data gets retained for some time

## External Table

▶ ▾ ✓ May 04, 2025 (5s) 18

```
CREATE TABLE sparksql_cata.sparksql_schema.order_ext
LOCATION 'abfss://sparksql@databricksete.dfs.core.windows.net/source/orders_ext'
AS
SELECT * FROM orders_temp;
```

▶ 📄 \_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num\_affected\_rows: long, num\_inserted\_rows: long]

Table ▾ +

1 <sup>2</sup> <sub>3</sub> num_affected_rows	1 <sup>2</sup> <sub>3</sub> num_inserted_rows
---	---

▶ ✓ May 04, 2025 (1s) 4

```
%python
df_test = spark.sql('''SELECT
| *
| FROM
|   sparksql_cata.sparksql_schema.order_ext
| WHERE
|   product_category = "Fashion";''')
```

▶ 📄 df\_test: pyspark.sql.connect.dataframe.DataFrame = [order\_id: integer, user\_id: string ... 8 more fields]

## SQL Warehouse → compute built for SQL workloads

The screenshot shows the Databricks web interface. On the left is the 'Catalog' sidebar with a search bar and a tree view of the database structure. The tree includes 'My organization', 'system', '\_databricks\_internal', 'databricks\_cata', 'main', 'my\_ms\_sql\_server\_catalog', 'sparksql\_cata', 'default', 'information\_schema', 'sparksql\_schema' (highlighted with a red line), 'order\_ext' (highlighted with a red line), 'order\_man', 'Delta Shares Received', and 'samples'. The main area displays a SQL query titled 'query\_1\_aggregations'. The query is: 

```
1 SELECT
2   month(order_date) as Order_Month,
3   product_category,
4   count(order_id) as total_orders
5 FROM
6   sparksql_cata.sparksql_schema.order_ext
7 GROUP BY
8   Order_Month, product_category;
```

 Below the query, the 'Raw results' tab is active, showing a table with 5 rows and 3 columns: 'Order\_Month', 'product\_category', and 'total\_orders'. The table data is as follows:

	Order_Month	product_category	total_orders
1	5	Electronics	27
2	5	Fashion	21
3	4	Books	174
4	5	Books	19
5	4	Fashion	167

Sequence of execution : from → where → groupby → select

This screenshot shows a different SQL query in the Databricks interface. The query is: 

```
1 SELECT
2   month(order_date) as Order_Month,
3   product_category,
4   count(order_id) as total_orders
5 FROM
6   sparksql_cata.sparksql_schema.order_ext
7 GROUP BY
8   Order_Month, product_category
9 ORDER BY Order_Month asc, total_orders desc;
```

 The 'Raw results' tab is active, showing a table with 10 rows and 3 columns: 'Order\_Month', 'product\_category', and 'total\_orders'. The table data is as follows:

	Order_Month	product_category	total_orders
1	4	Electronics	196
2	4	Kitchen	177
3	4	Books	174
4	4	Home Decor	172
5	4	Fashion	167
6	5	Electronics	27
7	5	Kitchen	26
8	5	Fashion	21
9	5	Home Decor	21
10	5	Books	19

## Subquery

```
1 SELECT * FROM
2 (
3 SELECT
4     month(order_date) as Order_Month,
5     product_category,
6     count(order_id) as total_orders
7 FROM
8     sparksql_cata.sparksql_schema.order_ext
9 GROUP BY
10     Order_Month, product_category
11 ORDER BY Order_Month asc, total_orders desc
12 ) tbl1
13 WHERE
14     product_category = 'Home Decor'
```

## Conditional statements

```
1 SELECT
2     *,
3     CASE
4         WHEN (payment_method LIKE '%Card%') AND (order_status IN ('Cancelled','Returned')) THEN 'CARD'
5         WHEN (payment_method IN ('Paypal','UPI')) AND (order_status IN ('Cancelled','Returned')) THEN 'CASH'
6         ELSE 'No Value' END Payment_Flag
7 FROM
8     sparksql_cata.sparksql_schema.order_ext
9
```

Raw results

	product_name	quantity	price_per_unit	payment_method	order_status	Payment_Flag
1	Sneakers	2	58.53	PayPal	Cancelled	No Value
2	T-Shirt	3	83.76	UPI	Returned	CASH
3	Sunglasses	2	78.85	PayPal	Processing	No Value
4	Sunglasses	5	46.49	PayPal	Delivered	No Value
5	Photo Frame	2	78.61	PayPal	Returned	No Value

## Subqueries are alternatives to CTE

```
1 WITH tbl1
2 (
3 SELECT
4     *,
5     CASE
6         WHEN (payment_method LIKE '%Card%') AND (order_status IN ('Cancelled','Returned')) THEN 'CARD'
7         WHEN (payment_method IN ('Paypal','UPI')) AND (order_status IN ('Cancelled','Returned')) THEN 'CASH'
8         ELSE 'No Value' END Payment_Flag
9 FROM
10     sparksql_cata.sparksql_schema.order_ext
11 )
12 SELECT * FROM tbl1
13 WHERE Payment_Flag='CARD'
14
```

## Hierarchical CTEs

```
Run (1000) databricks_ete.Select schema sql_compute Serverless 2XS Save*

1 WITH tb12
2 (
3   WITH tb11
4   (
5     SELECT
6       *,
7       CASE
8         WHEN (payment_method LIKE '%Card%') AND (order_status IN ('Cancelled','Returned')) THEN 'CARD'
9         WHEN (payment_method IN ('Paypal','UPI')) AND (order_status IN ('Cancelled','Returned')) THEN 'CASH'
10        ELSE 'No Value' END Payment_Flag
11    FROM
12      sparksql_cata.sparksql_schema.order_ext
13  )
14  SELECT * FROM tb11
15  WHERE Payment_Flag='CARD'
16 )
17 SELECT * FROM tb12
18 WHERE product_category = 'Home Decor'
```

## Window Functions

```
1 SELECT
2   price_per_unit,
3   Rank() over(order by price_per_unit desc) as rank,
4   Dense_Rank() over(order by price_per_unit desc) as dense_rank,
5   Row_Number() over(order by price_per_unit desc) as row_number
6 FROM
7   sparksql_cata.sparksql_schema.order_ext
```

30	97.17	30	30	30
31	97.14	31	31	31
32	96.98	32	32	32
33	96.98	32	32	33
34	96.77	34	33	34
35	96.74	35	34	35
36	96.59	36	35	36
37	96.57	37	36	37
38	96.5	38	37	38
39	96.48	39	38	39
40	96.47	40	39	40

```
Run (1000) databricks_ete.Select schema sql_compute Serverless 2XS Save* Schedule Share

1 SELECT
2   *,
3   SUM(price_per_unit) OVER(order by price_per_unit rows between unbounded preceding and UNBOUNDED FOLLOWING) AS total_order_value
4 FROM
5   sparksql_cata.sparksql_schema.order_ext
```

```
SELECT
*,
SUM(price_per_unit) OVER(order by price_per_unit rows between unbounded preceding and UNBOUNDED FOLLOWING) AS total_order_value,
SUM(price_per_unit) OVER(order by price_per_unit rows between unbounded preceding and CURRENT ROW) AS running_total
FROM
sparksql_cata.sparksql_schema.order_ext
```

Raw results

	quantity	1.2 price_per_unit	A <sub>C</sub> payment_method	A <sub>C</sub> order_status	1.2 total_order_value	1.2 running_total
1	4	10.03	Credit Card	Cancelled	55205.360000000007	10.03
2	2	10.16	PayPal	Cancelled	55205.360000000007	20.189999999999998
3	5	10.46	Credit Card	Returned	55205.360000000007	30.65
4	4	10.51	Debit Card	Processing	55205.360000000007	41.16
5	3	10.59	Credit Card	Returned	55205.360000000007	51.75
6	2	10.59	Debit Card	Processing	55205.360000000007	62.34
7	3	10.6	UPI	Processing	55205.360000000007	72.94

## UPSERT - MERGE

May 05, 2025 (12s) 2

```
%python
df = spark.read.table("sparksql_cata.sparksql_schema.order_man")
```

df: pyspark.sql.connect.dataframe.DataFrame = [order\_id: integer, user\_id: string ... 8 more fields]

May 05, 2025 (2s) 3

```
%python
df.createOrReplaceTempView("order_source")
```

May 05, 2025 (2s) 4

```
%python
spark.sql('SELECT * FROM {orders_temp}',orders_temp = df).display()
```

The last 2 commands are equivalent

▶

✓

May 05, 2025 (10s)

5

```

MERGE INTO sparksql_cata.sparksql_schema.order_ext trg
USING order_source src
ON trg.order_id = src.order_id
WHEN MATCHED THEN
UPDATE SET *
WHEN NOT MATCHED THEN
INSERT *

```

▶

\_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num\_affected\_rows: long, num\_updated\_rows: long ... 2 more fields]

Table

+

	12 num_affected_rows	12 num_updated_rows	12 num_deleted_rows	12 num_inserted_rows
1	1000	1000	0	0

## FUNCTIONS

2 types → 1) User Defined Scalar Function 2) User Defined Table Function  
Register Functions in UC

▼

⋮

FUNCTIONS

### SCALAR FUNCTION

▶

✓

May 05, 2025 (9s)

3

```

CREATE OR REPLACE FUNCTION sparksql_cata.sparksql_schema.discount_price(p_price DECIMAL(10,2))
RETURNS DECIMAL(10,2)
LANGUAGE SQL
RETURN p_price * 0.90

```

▶

✓

May 05, 2025 (5s)

4

```

SELECT price_per_unit, sparksql_cata.sparksql_schema.discount_price(price_per_unit) FROM sparksql_cata.sparksql_schema.order_ext

```

▶

\_sqldf: pyspark.sql.connect.dataframe.DataFrame = [price\_per\_unit: double, sparksql\_cata.sparksql\_schema.discount\_price(price\_per\_unit): decimal(10,2)]

Table

+

	12 price_per_unit	.00 sparksql_cata.sparksql_schema.discount_price(price_per_unit)
1	58.53	52.68
2	83.76	75.38
3	78.85	70.97
4	46.49	41.84
5	78.61	70.75
6	53.51	48.16
7	12.71	11.44
8	46.6	41.94
9	35.87	32.28
10	30.95	27.86
11	18.79	16.91
12	69.14	62.23
13	90.64	81.58
14	93.91	84.52
15	61	54.90



## TABLE FUNCTIONS - UDTF

6

```
CREATE OR REPLACE FUNCTION sparksql_cata.sparksql_schema.discount_price(p_category STRING)
RETURNS TABLE
LANGUAGE SQL
RETURN
(SELECT * FROM sparksql_cata.sparksql_schema.order_ext WHERE product_category = p_category)
```

7

```
SELECT * FROM sparksql_cata.sparksql_schema.discount_price('Home Decor')
```

\_sqlidf: pyspark.sql.connect.dataframe.DataFrame = [order\_id: integer, user\_id: string ... 8 more fields]

	order_id	user_id	order_date	product_id	product_category	product_name	quantity	price_per_unit	payment_method
1	1005	U003	2025-04-19	P988	Home Decor	Photo Frame	2	78.61	PayPal
2	1007	U129	2025-04-23	P786	Home Decor	Wall Clock	5	12.71	Credit Card
3	1008	U102	2025-04-15	P101	Home Decor	Photo Frame	1	46.6	Debit Card
4	1034	U016	2025-04-23	P948	Home Decor	Wall Clock	3	54.5	UPI
5	1035	U150	2025-04-10	P242	Home Decor	Cushion Cover	2	34.04	Credit Card
6	1036	U171	2025-05-02	P585	Home Decor	Cushion Cover	2	17.22	Credit Card
7	1040	U172	2025-04-08	P644	Home Decor	Wall Art	2	97.17	Debit Card
8	1051	U049	2025-04-25	P219	Home Decor	Wall Art	3	78.9	UPI
9	1058	U090	2025-04-16	P250	Home Decor	Wall Clock	5	72.65	UPI
10	1061	U069	2025-04-06	P463	Home Decor	Wall Clock	1	96.35	PayPal
11	1063	U187	2025-05-03	P772	Home Decor	Wall Art	2	46.91	Debit Card
12	1064	U152	2025-04-28	P133	Home Decor	Wall Clock	3	56.43	UPI
13	1066	U128	2025-04-15	P639	Home Decor	Cushion Cover	4	20.13	Credit Card
14	1076	U141	2025-04-13	P898	Home Decor	Lamp	3	41.6	UPI

## Dynamic Data Masking

Prerequisite: need admin rights

Microsoft Azure databricks

Search data, notebooks, recent, and more... CTRL + P

Workspace settings > Identity and access >

Groups

Filter groups 3 total

Name	Members	Source
New Users	1	Account
admins	2	System
users	3	System

Previous Next 20 / page

Settings

- Workspace admin
- Appearance
- Identity and access
- Security
- Compute
- Development
- Notifications
- Advanced
- User
  - Profile
  - Preferences
  - Developer

Setting

Add group

Add yourself as an admin





+ Code + Text ...



# DYNAMIC DATA MASKING

## MASK FUNCTION

▶ ✓ May 05, 2025 (3s) 3

```
CREATE OR REPLACE FUNCTION sparksql_cata.sparksql_schema.dynamic_mask(p_user_id STRING)
RETURN
CASE WHEN is_account_group_member('admin') THEN p_user_id ELSE '*****' END;
```

Masking PII (Personal identifiable information) → used\_id  
is\_account\_group\_member() is an inbuilt function

## APPLYING MASK FUNCTION TO THE COLUMN - user\_id

▶ ✓ May 05, 2025 (2s) 5

```
ALTER TABLE sparksql_cata.sparksql_schema.order_ext
ALTER COLUMN user_id SET MASK sparksql_cata.sparksql_schema.dynamic_mask;
```

▶ \_sqldf: pyspark.sql.connect.dataframe.DataFrame = [order\_id: integer, user\_id: string ... 8 more fields]

Table ▾		+							🔍 🔍 📄 📄	
	order_id	user_id	order_date	product_id	product_category	product_name	quantity	price		
1	1001	*****	2025-04-20	P940	Fashion	Sneakers	2			
2	1002	*****	2025-04-16	P794	Fashion	T-Shirt	3			
3	1003	*****	2025-04-18	P326	Fashion	Sunglasses	2			
4	1004	*****	2025-04-10	P574	Fashion	Sunglasses	5			
5	1005	*****	2025-04-19	P988	Home Decor	Photo Frame	2			
6	1006	*****	2025-04-15	P328	Kitchen	Knife Set	4			
7	1007	*****	2025-04-23	P786	Home Decor	Wall Clock	5			

**Row Level Security** → restricting access to the users/what users should have access to what data  
(Normally applied in PowerBI but can be used in ADB)

⋮

ROW LEVEL SECURITY

Markdown

Mapping Table

▶

✓ May 05, 2025 (14s)

3

CREATE TABLE sparksql\_cata.sparksql\_schema.map\_table  
(  
  payment\_category STRING,  
  email STRING  
)

▶

✓ May 05, 2025 (22s)

4

SELECT \* FROM sparksql\_cata.sparksql\_schema.order\_ext

▶

\_sqldf: pyspark.sql.connect.dataframe.DataFrame = [order\_id: integer, user\_id: string ... 8 more fields]

Table

▼

+

	123 order_id	A0 user_id	📅 order_date	A0 product_id	A0 product_category	A0 product_name	123 quantity	1.2 price_per_unit	A0 payment_mett
1	1001	*****	2025-04-20	P940	Fashion	Sneakers	2	58.53	PayPal
2	1002	*****	2025-04-16	P794	Fashion	T-Shirt	3	83.76	UPI
3	1003	*****	2025-04-18	P326	Fashion	Sunglasses	2	78.85	PayPal
4	1004	*****	2025-04-10	P574	Fashion	Sunglasses	5	46.49	PayPal
5	1005	*****	2025-04-19	P988	Home Decor	Photo Frame	2	78.61	PavPal

▶

▼

✓ May 05, 2025 (4s)

6

INSERT INTO sparksql\_cata.sparksql\_schema.map\_table  
VALUES  
(  
  'Credit Card', 'anshlambaaz@gmail.com'),  
  'Debit Card', 'anshlambaaz@gmail.com'),  
  'PayPal', 'newuser@anshlambaazgmail.onmicrosoft.com'),  
  'UPI', 'newuser@anshlambaazgmail.onmicrosoft.com')

▶

\_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num\_affected\_rows: long, num\_inserted\_rows: lor

Table

▼

+

	123 num_affected_...	123 num_inserted_rows
1	4	4

▶

▼

✓ Just now (4s)

6

SELECT \* FROM sparksql\_cata.sparksql\_schema.map\_table  
WHERE email = current\_user()

Table

▼

+

	A0 payment_category	A0 email
1	Credit Card	anshlambaaz@gmail.com
2	Debit Card	anshlambaaz@gmail.com

## Mapping Table Testing

▶ ✓ May 05, 2025 (3s) 7

```
SELECT * FROM sparksql_cata.sparksql_schema.map_table
WHERE email = current_user()
AND payment_category = 'Credit Card'
```

▶ \_sqldf: pyspark.sql.connect.dataframe.DataFrame = [payment\_category: string, email: string]

Table ▾ +

	<sup>A</sup> <sub>C</sub> payment_category	<sup>A</sup> <sub>C</sub> email
1	Credit Card	anshlambaaz@gmail.com

▶ ▾ ✓ Just now (4s) 6

```
SELECT * FROM sparksql_cata.sparksql_schema.map_table
WHERE email = current_user()
AND payment_category = 'UPI' I
```

> [See performance \(1\)](#)

Table ▾ +

	<sup>A</sup> <sub>C</sub> payment_category	<sup>A</sup> <sub>C</sub> email
--	--	---------------------------------

No data cause UPI data should be visible to the other user

## Converting Mapping Table into a Boolean

▶ ▾ ✓ May 05, 2025 (3s) 9

```
SELECT EXISTS
(
  SELECT * FROM sparksql_cata.sparksql_schema.map_table
  WHERE email = current_user()
  AND payment_category = 'UPI'
)
```

▶ \_sqldf: pyspark.sql.connect.dataframe.DataFrame = [EXISTS(SELECT\*FROMsparksql\_cata.sparksql\_schema.map\_tableWHEREemail=current\_user()AND

Table ▾ +

	<sup>A</sup> <sub>C</sub> EXISTS(SELECT*FROMsparksql_cata.sparksql_schema.map_tableWHEREemail=current_user()ANDpayment_category='UPI')
1	false

## CONVERT INTO A BOOLEAN FUNCTION

```
▶ ✓ May 05, 2025 (3s) 11

CREATE OR REPLACE FUNCTION sparksql_cata.sparksql_schema.rowlevel_security(p_payment_method STRING)
RETURNS BOOLEAN
LANGUAGE SQL
RETURN
(
  EXISTS
  (
    SELECT * FROM sparksql_cata.sparksql_schema.map_table
    WHERE email = current_user()
    AND payment_category = p_payment_method
  )
)
```

## APPLYING RLS FUNCTION TO THE COLUMN

```
< > + Code + Te

▶ ✓ May 05, 2025 (3s) 13

ALTER TABLE sparksql_cata.sparksql_schema.order_ext
SET ROW FILTER sparksql_cata.sparksql_schema.rowlevel_security ON (payment_method)
```

## TESTING

```
▶ ✓ Just now (5s) 15

SELECT * FROM sparksql_cata.sparksql_schema.order_ext
```

	1.2 order_id	A. user_id	order_date	A. product_id	A. product_category	A. product_name	1.2
1	1006	*****	2025-04-15	P328	Kitchen	Knife Set	
2	1007	*****	2025-04-23	P786	Home Decor	Wall Clock	
3	1008	*****	2025-04-15	P101	Home Decor	Photo Frame	
4	1009	*****	2025-04-04	P610	Kitchen	Toaster	
5	1010	*****	2025-04-29	P354	Kitchen	Microwave	
6	1012	*****	2025-04-24	P315	Fashion	Sunglasses	
7	1013	*****	2025-05-03	P516	Fashion	Sneakers	
8	1014	*****	2025-04-18	P111	Books	Data Engineering 101	
9	1018	*****	2025-04-15	P650	Electronics	Wireless Mouse	
10	1019	*****	2025-05-03	P973	Kitchen	Blender	
11	1025	*****	2025-04-23	P311	Kitchen	Cutting Board	
12	1026	*****	2025-04-05	P793	Fashion	Sneakers	
13	1032	*****	2025-04-11	P895	Kitchen	Cutting Board	
14	1033	*****	2025-04-07	P518	Kitchen	Microwave	
15							

	t_category	A. product_name	1.2 quantity	1.2 price_per_unit	A. payment_method	A. order_status
1		Knife Set	4	53.51	Credit Card	Returned
2	or	Wall Clock	5	12.71	Credit Card	Returned
3	or	Photo Frame	1	46.6	Debit Card	Cancelled
4		Toaster	4	35.87	Credit Card	Processing
5		Microwave	1	30.95	Credit Card	Processing
6		Sunglasses	5	69.14	Credit Card	Processing
7		Sneakers	5	90.64	Credit Card	Cancelled
8		Data Engineering 101	1	93.91	Credit Card	Cancelled
9		Wireless Mouse	2	18.72	Debit Card	Processing
10		Blender	3	35.68	Debit Card	Delivered
11		Cutting Board	4	16.8	Debit Card	Returned
12		Sneakers	4	77.54	Credit Card	Processing
13		Cutting Board	2	29.79	Credit Card	Cancelled
14		Microwave	4	34.77	Credit Card	Processing
15						

So whats happening bts : when we are querying the table, we have applied row filter on payment\_method column, simply pass all the values to the rowlevel\_security function, this function will take all the values of payment\_method & check the current user and privilege of that user.

# DML OPERATIONS

▶

✓ Just now (4s)

```
UPDATE sparksql_cata.sparksql_schema.order_man
SET product_category = 'GenZ Fashion'
WHERE product_category = 'Fashion'
```

>

[See performance \(1\)](#)

Table ▼

+

	1 <sup>2</sup> 3 num_affected_rows	
1	188	+

```
DESCRIBE sparksql_cata.sparksql_schema.order_man
```

>

[See performance \(1\)](#)

I

Table ▼

+

	A <sup>B</sup> C col_name	A <sup>B</sup> C data_type	A <sup>B</sup> C comment
1	order_id	int	null
2	user_id	string	null
3	order_date	date	null
4	product_id	string	null
5	product_category	string	null
6	product_name	string	null
7	quantity	int	null
8	price_per_unit	double	null
9	payment_method	string	null
10	order_status	string	null

▶

✓ Just now (2s)

5

```
DESCRIBE EXTENDED sparksql_cata.sparksql_schema.order_man
```

>

[See performance \(1\)](#)

9	payment_method	string
10	order_status	string
11		
12	# Delta Statistics Columns	
13	Column Names	quantity, order_id, price_per_unit, user_id, payment_method, order_status, product_name, order_date, product_id, product_
14	Column Selection Method	first-32
15		
16	# Detailed Table Information	+
17	Catalog	sparksql_cata
18	Database	sparksql_schema
19	Table	order_man
20	Created Time	Sun May 04 17:44:17 UTC 2025

Delta Tables calculate the statistics of first 32 columns

To see performance

te ago

This result is stored as `_sqlidf` and can be used in other Python and SQL cells.

1 minute ago (2s) 5

DESCRIBE EXTENDED sparksql\_cata.sparksql\_schema.order\_man

Hide performance (1)

Statement	Started At	Duration
DESCRIBE EXTENDED sparksql	May 04, 2025, 05:58 PM	232 ms

Table	col_name	data_type
# Detailed Table Information		
Catalog		sparksql_cata
Database		sparksql_schema
Table		order_man
Created Time		Sun May 04 17:44:17 UTC 2025

Ansri Lambda - Serverless compute

Files read: 0, Files written: 0, See query profile >

See longest operations for this query

Query wall-clock duration ⓘ

Total wall-clock duration: 232 ms

Optimizing query & pruning files ⓘ: 63%, 146 ms

Executing ⓘ: 37%, 86 ms

Start time: May 04, 2025, 05:58:19 PM GMT-03:00

End time: May 04, 2025, 05:58:19 PM GMT-03:00

Result fetching by client ⓘ: 38 ms

Query Source

> 7\_sparksql / Cell(ID: 613...00060601665)

Aggregated task time ⓘ

Tasks total time

Tasks time in Photon



Just now (8s) 11

RESTORE sparksql\_cata.sparksql\_schema.order\_man TO VERSION AS OF 3

See performance (1) Optimize

Table	table_size_after_restore	num_of_files_after_restore	num_removed_files	num_restored_files	remove
1	20107	1	0	1	

## Different approach

Waiting 2

```
CREATE TABLE sparksql_cata.sparksql_schema.order_new
LOCATION 'abfss://sparksql@databricksete.dfs.core.windows.net/source/orders_new'
AS SELECT * FROM sparksql_cata.sparksql_schema.order_man
```

Just now (4s) 14

DESCRIBE HISTORY delta.`abfss://sparksql@databricksete.dfs.core.windows.net/source/orders\_new`

See performance (1) Optimize

Table	version	timestamp	userId	userName	operation	operationParam
1	0	2025-05-04T21:08:36.000+00:...	13286368820035...	anshlambaaz@gmail.com	CREATE TABLE AS SELECT	> [{"partitionBy": ""}]

Interrupt 00:01

```
UPDATE sparksql_cata.sparksql_schema.order_new
SET product_category = 'GenZ Fashion'
WHERE product_category = 'Fashion'
```

Just now (4s) 14

DESCRIBE HISTORY delta.`abfss://sparksql@databricksete.dfs.core.windows.net/source/orders\_new`

See performance (1) Optimize

Table	version	timestamp	userId	userName	operation	operationParam
1	1	2025-05-04T21:09:49.000+00:...	13286368820035...	anshlambaaz@gmail.com	UPDATE	> [{"predicate": "\n(f"}]
2	0	2025-05-04T21:08:36.000+00:...	13286368820035...	anshlambaaz@gmail.com	CREATE TABLE AS SELECT	> [{"partitionBy": ""}]

▶

✓

Just now (6s)

15

SQL

✦

⌵

⋮

🗑

RESTORE delta.`abfss://sparksql@databricksete.dfs.core.windows.net/source/orders\_new`  
TO VERSION AS OF 0

> [See performance \(1\)](#)

Optimize

Table ▾

+

🔍

🔼

📊

🗑

	<sup>1</sup> <sub>3</sub> table_size_after_restore	<sup>1</sup> <sub>3</sub> num_of_files_after_restore	<sup>1</sup> <sub>3</sub> num_removed_files	<sup>1</sup> <sub>3</sub> num_restored_files	<sup>1</sup> <sub>3</sub> remove
1	20101	1	0	0	