

Data Sheet

What is Dynamic View Access Control in Databricks?

Implementation with
Practical Examples

5201 GREAT AMERICAN PARKWAY, SUITE 320

SANTA CLARA, CA 95054

Tel: (855) 695-8636

E-mail: info@lumendata.com

Website: www.lumendata.com

Dynamic View Access Control (DVAC) in Databricks is a powerful feature that allows organizations to implement row-level, column-level, and dynamic masking security in their data lake without compromising performance or maintainability.

This data sheet explores how DVAC solves common data access challenges and demonstrates its implementation with practical examples.

Understanding Dynamic View Access

With automatic view creation resulting in filtering of data based on user identity and attributes, DVAC aids fine-grained access control at the row level. Unlike managing complex ACLs or creating several copies of tables for different groups of users, with DVAC, dynamic security filters are applied during user queries on the underlying data.

Note: For using dynamic views, the workspace must be enabled for serverless compute. This is because data filtering functionality which enables dynamic views runs on serverless compute.

Key Challenges Faced by Organizations that Can Be Solved by DVAC

- In a traditional approach, table-level permissions are usually too coarse-grained for complex organisations.
- Previous solutions involving user-based filters could significantly impact query performance.
- Use of multiple tables or views per group brings complexity into operations.
- Multiple copies of data for different access levels can lead to synchronization issues.

Workflow Improvements

- Simplified access management through centralized policies.
- Reduced storage costs by eliminating duplicate data sets.
- Optimized filtering to enhance query processing performance.
- With consistent access patterns, audit capability is improved.

Advanced Permission Controls in Dynamic Views

Column-Level Permissions

Column-level permissions allow you to control the access of certain columns to specific groups of users.

Row-Level Permission

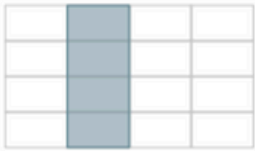
Row-level permissions enable fine-grained control over which records users can access. This includes security at the level of data access. No performance effect as filtering happens at the query planning time.

Data Masking

Data masking enables protection for sophisticated data with analytical capabilities intact:

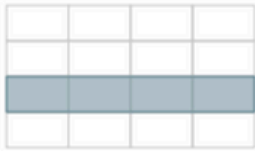
Limit access to columns

Omit column values from output



Limit access to rows

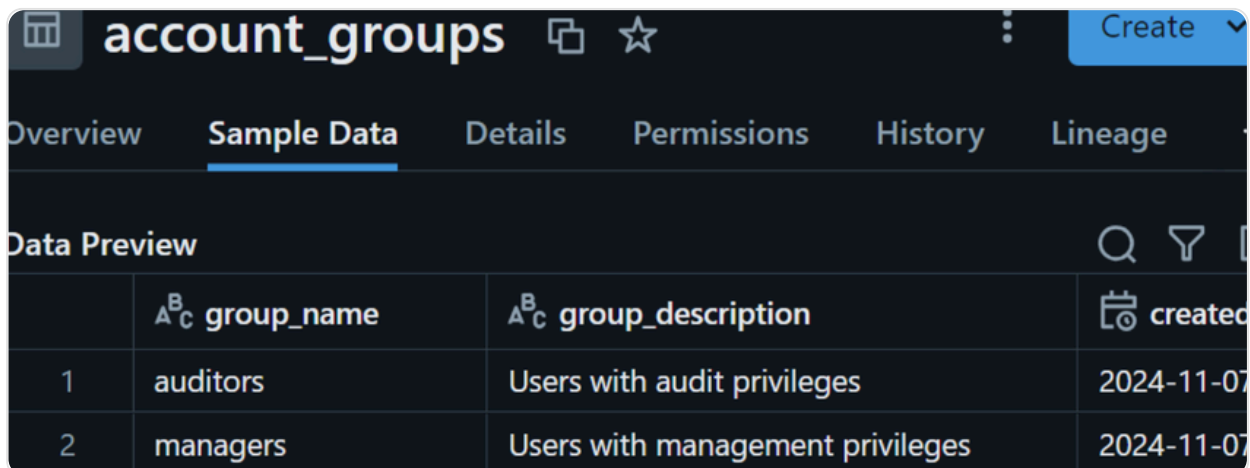
Omit rows from output



Implementation Examples

Let's explore how to implement DVAC using both PySpark and SQL approaches.

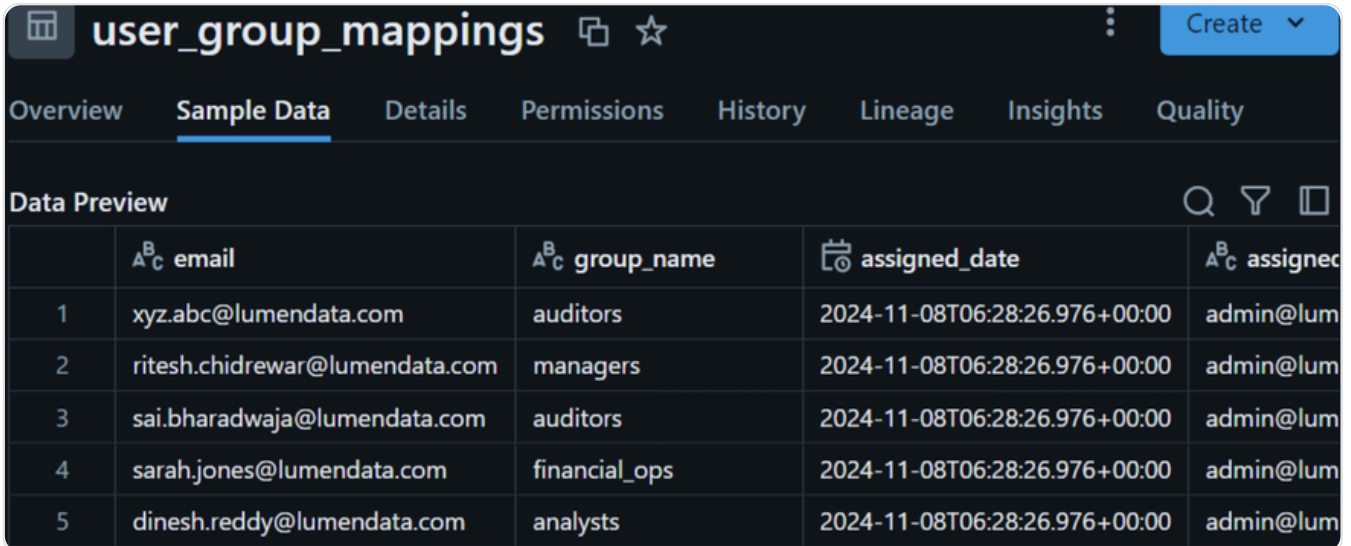
1. Set up the necessary tables for group management.



	^A _C group_name	^A _C group_description	created
1	auditors	Users with audit privileges	2024-11-07
2	managers	Users with management privileges	2024-11-07

What is Dynamic View Access Control in Databricks?

The '**user_group_mappings**' table contains information about user assignments to groups. It includes details such as the user's email address, the group name, the date of assignment, and the person who assigned the user to the group.



The screenshot shows the Databricks interface for the 'user_group_mappings' table. The 'Sample Data' tab is selected, displaying a data preview with 5 rows. The columns are 'email', 'group_name', 'assigned_date', and 'assigned_by'.

	^A _C email	^A _C group_name	^A _C assigned_date	^A _C assigned_by
1	xyz.abc@lumendata.com	auditors	2024-11-08T06:28:26.976+00:00	admin@lum
2	ritesh.chidrewar@lumendata.com	managers	2024-11-08T06:28:26.976+00:00	admin@lum
3	sai.bharadwaja@lumendata.com	auditors	2024-11-08T06:28:26.976+00:00	admin@lum
4	sarah.jones@lumendata.com	financial_ops	2024-11-08T06:28:26.976+00:00	admin@lum
5	dinesh.reddy@lumendata.com	analysts	2024-11-08T06:28:26.976+00:00	admin@lum

2. Create **is_account_group_member** function for checking the user in groups.

```
# 5. Create a function is_account_group_member
spark.sql("""
CREATE OR REPLACE FUNCTION ld_analytics.ritesh_demo.is_account_group_member
(input_group_name STRING)
RETURNS BOOLEAN
RETURN EXISTS (
  SELECT 1
  FROM user_group_mappings
  WHERE email = current_user()
  AND group_name = input_group_name
)
""")
```

Sample data for testing in table **financial_transactions** table.

	transaction_id	customer_email	amount	transaction_date	sensitive_info
1	T1001	customer1@email.com	500000.00	2024-11-07	SSN: 123-45-6789
2	T1002	customer2@email.com	1500000.00	2024-11-07	SSN: 234-56-7890
3	T1003	customer3@email.com	750000.00	2024-11-07	SSN: 345-67-8901
4	T1004	customer4@email.com	2000000.00	2024-11-07	SSN: 456-78-9012
5	T1005	customer5@email.com	100000.00	2024-11-07	SSN: 567-89-0123

Created secure financial view to restrict data access to users as per their group.

Security Features:

- Email masking for non-analysts.
- Amount restrictions for non-managers.
- Sensitive info protection.

```
--  
62 # 7. Create secure view using group membership  
63 spark.sql("""  
64 CREATE OR REPLACE VIEW ld_analytics.ritesh_demo.secure_financial_view AS  
65 SELECT  
66     transaction_id,  
67     -- Only analysts can see customer email  
68     CASE WHEN is_account_group_member('analysts')  
69         THEN customer_email  
70         ELSE 'REDACTED'  
71     END AS customer_email,  
72     -- Managers can see all amounts, others only see if under 1M  
73     CASE WHEN is_account_group_member('managers')  
74         THEN amount  
75         WHEN amount <= 1000000 THEN amount  
76         ELSE NULL  
77     END AS amount,  
78     transaction_date,  
79     -- Only auditors can see sensitive info  
80     CASE WHEN is_account_group_member('auditors')  
81         THEN sensitive_info  
82         ELSE 'REDACTED'  
83     END AS sensitive_info  
84 FROM ld_analytics.ritesh_demo.financial_transactions
```

Testing Functions:

- test_view_as_user: Tests view access as different users.
- run_tests: Runs all tests.
- cleanup: Removes all created objects.

```
20 hours ago (<1s) 11 Python
1 # 8. Function to test different user perspectives
2 def test_view_as_user(email):
3     spark.sql(f"SET spark.sql.user.name = '{email}'")
4     # Query the secure view
5     print(f"\nViewing as {email}:")
6     return spark.sql("SELECT * FROM ld_analytics.ritesh_demo.secure_financial_view").show(truncate=False)
7
8 # 9. Test the view with different users
9 def run_tests():
10    # Test as auditor
11    test_view_as_user('ritesh.chidrewar@lumendata.com') # Manager
12
13    # Test as manager only
14    test_view_as_user('sai.bharadwaja@lumendata.com') # Auditors only
15
16    # Test as analyst
17    test_view_as_user('dinesh.reddy@lumendata.com') # Analyst only
```

```
1 # 11. Clean up the tables and views
2 def cleanup():
3     spark.sql("DROP TABLE IF EXISTS ld_analytics.ritesh_demo.account_groups")
4     spark.sql("DROP TABLE IF EXISTS ld_analytics.ritesh_demo.user_group_mappings")
5     spark.sql("DROP TABLE IF EXISTS ld_analytics.ritesh_demo.financial_transactions")
6     spark.sql("DROP VIEW IF EXISTS ld_analytics.ritesh_demo.secure_financial_view")
7     spark.sql("DROP FUNCTION IF EXISTS ld_analytics.ritesh_demo.is_account_group_member")
```

Test Results

Basic Fields (No Restrictions):

transaction_id, transaction_date,

- These fields are visible to all users.
- No CASE statement needed.
- No sensitive information contained.

Email Protection (Binary Access):

- Only Analysts see the actual email.
- Everyone else sees 'REDACTED'.
- Binary access control (either full access or no access).

Amount Protection (Tiered Access):

Three-tier access system:

1. Managers see all amounts.
2. Non-managers see amounts \leq \$1M.
3. Amounts $>$ \$1M are hidden (NULL) for non-managers.

Uses cascading CASE statement for multiple conditions

Sensitive Info Protection (Binary Access):

- Similar to email protection.
- Only auditors see sensitive info.
- Everyone else sees 'REDACTED'.

```
Just now (1s) 13 Python
1 # Run all tests
2 run_tests()
(4) Spark Jobs
Viewing as ritesh.chidrewar@lumendata.com as Manager:
+-----+-----+-----+-----+-----+
|transaction_id|customer_email|amount      |transaction_date|sensitive_info|
+-----+-----+-----+-----+-----+
|T1001         |REDACTED      |500000.00  |2024-11-08      |REDACTED      |
|T1002         |REDACTED      |1500000.00 |2024-11-08      |REDACTED      |
|T1003         |REDACTED      |750000.00  |2024-11-08      |REDACTED      |
|T1004         |REDACTED      |2000000.00 |2024-11-08      |REDACTED      |
|T1005         |REDACTED      |100000.00  |2024-11-08      |REDACTED      |
+-----+-----+-----+-----+-----+
```

1. Managers See:

- All transaction IDs.
- Redacted emails.
- All amounts.
- All dates.
- Redacted sensitive info.

```
2 run_tests()
(5) Spark Jobs
Viewing as sai.bharadwaja@lumendata.com as Auditors:
```

transaction_id	customer_email	amount	transaction_date	sensitive_info
T1001	REDACTED	500000.00	2024-11-08	SSN: 123-45-6789
T1002	REDACTED	NULL	2024-11-08	SSN: 234-56-7890
T1003	REDACTED	750000.00	2024-11-08	SSN: 345-67-8901
T1004	REDACTED	NULL	2024-11-08	SSN: 456-78-9012
T1005	REDACTED	100000.00	2024-11-08	SSN: 567-89-0123

2. Auditors See:

- All transaction IDs.
- All amounts (based on manager status).
- All dates.
- All sensitive info.

```
1 # Run all tests
2 run_tests()
(5) Spark Jobs
Viewing as dinesh.reddy@lumendata.com as Analysts:
```

transaction_id	customer_email	amount	transaction_date	sensitive_info
T1001	customer1@email.com	500000.00	2024-11-08	REDACTED
T1002	customer2@email.com	NULL	2024-11-08	REDACTED
T1003	customer3@email.com	750000.00	2024-11-08	REDACTED
T1004	customer4@email.com	NULL	2024-11-08	REDACTED
T1005	customer5@email.com	100000.00	2024-11-08	REDACTED

The view creates different "views" of the same data based on user group membership:

Regular Users/Analysts see:

- All transaction IDs.
- All emails.
- All dates.
- Redacted sensitive info.

Wrapping up

Dynamic View Access Control in Databricks offers strong security for modern data architectures. The combination of column-level permissions, row-level filtering, and data masking enables organizations to:

- Enable fine-grained access control without compromise.
- Perform optimized query execution performance with native filtering.
- Centralize access policy management.
- Implement, audit, and comply with regulatory requirements.

Interested in learning or understanding what Databricks features can work best for your organization? [Connect with LumenData today.](#)

Authors



Ritesh Chidrewar
Senior Consultant

About LumenData

LumenData is a leading provider of **Enterprise Data Management, Cloud & Analytics** solutions. We help businesses navigate their data visualization and analytics anxieties and enable them to accelerate their innovation journeys.

Founded in 2008, with locations in multiple countries, LumenData is privileged to serve over 100 leading companies. LumenData is **SOC2 certified** and has instituted extensive controls to protect client data, including adherence to GDPR and CCPA regulations.



Get in touch with us:
info@lumendata.com

Let us know what you need:
lumendata.com/contact-us

