

Portfolio Project:

Securing Data in a Data Warehouse

Project Overview

This project focused on securing a Microsoft Fabric data warehouse using various security features, including granular permissions, column-level security, row-level security, and dynamic data masking. The goal was to ensure that sensitive data is protected while allowing appropriate access to authorized users.

Objectives:

- 1. Create a Workspace:**
 - Set up a workspace with the Fabric trial enabled.
- 2. Create a Data Warehouse:**
 - Establish a data warehouse for testing security features.
- 3. Apply Dynamic Data Masking:**
 - Implement dynamic data masking rules on specific columns.
- 4. Apply Row-Level Security:**
 - Restrict access to rows based on user identity.
- 5. Implement Column-Level Security:**
 - Control access to specific columns in a table.
- 6. Configure SQL Granular Permissions:**
 - Use T-SQL to manage permissions at a granular level.

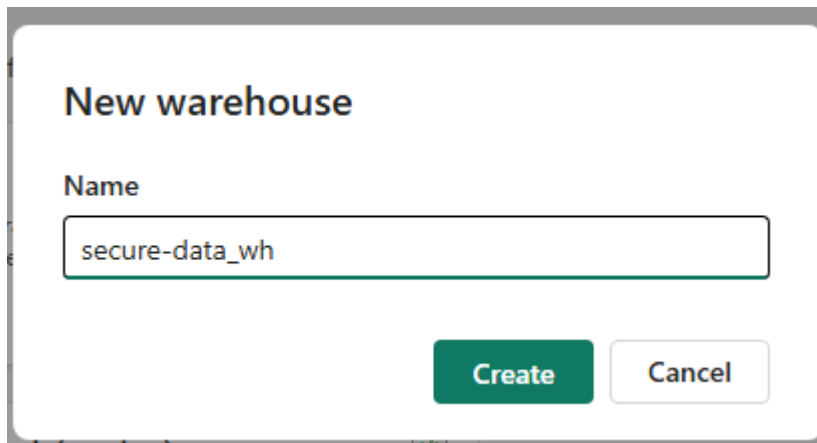
Experience

Create a Workspace

- Navigated to Microsoft Fabric Home and signed in.
- Selected Workspaces and created a new workspace.

Create a Data Warehouse

- Selected Create, then Warehouse, and gave it a unique name.



New warehouse

Name

secure-data_wh

Create Cancel

Apply Dynamic Data Masking

- Created a table with dynamic data masking:

The screenshot shows the Microsoft Fabric secure-data-wh interface. The left sidebar contains the Explorer pane with a tree view showing the database structure: Warehouses, secure-data-wh, Schemas, dbo, Tables, Customers, Views, Functions, Stored Procedures, INFORMATION_SCHEMA, queryinsights, sys, Security, and Queries. The main pane displays a SQL query titled 'SQL query 1' with the following code:

```

1 CREATE TABLE dbo.Customers
2 (
3     CustomerID INT NOT NULL,
4     FirstName varchar(50) MASKED WITH (FUNCTION = 'partial(1,"XXXXXXX",0)') NULL,
5     LastName varchar(50) NOT NULL,
6     Phone varchar(20) MASKED WITH (FUNCTION = 'default()') NULL,
7     Email varchar(50) MASKED WITH (FUNCTION = 'email()') NULL
8 );
9
10 INSERT dbo.Customers (CustomerID, FirstName, LastName, Phone, Email) VALUES
11 (29485, 'Catherine', 'Abel', '555-555-5555', 'catherine@adventure-works.com'),
12 (29486, 'Kim', 'Abercrombie', '444-444-4444', 'kim2@adventure-works.com'),
13 (29489, 'Frances', 'Adams', '333-333-3333', 'frances@adventure-works.com');
14
15 SELECT * FROM dbo.Customers;

```

Below the query, the 'Results' tab shows a table with 5 columns: CustomerID, FirstName, LastName, Phone, and Email. The table contains 3 rows of data:

	CustomerID	ABC FirstName	ABC LastName	ABC Phone	ABC Email
1	29485	Catherine	Abel	555-555-5555	catherine@adventure-works.com
2	29486	Kim	Abercrombie	444-444-4444	kim2@adventure-works.com
3	29489	Frances	Adams	333-333-3333	frances@adventure-works.com

The status bar at the bottom indicates 'Succeeded (16 sec: 435 ms)' and 'Copilot completions: On'.

- Tested with a user in the Viewer role to see masked data.

The screenshot shows the Microsoft Fabric secure-data-wh interface with the same SQL query as before. The 'Results' tab shows a table with 5 columns: CustomerID, FirstName, LastName, Phone, and Email. The table contains 3 rows of data, all of which are masked:

	CustomerID	ABC FirstName	ABC LastName	ABC Phone	ABC Email
1	29485	CXXXXXXX	Abel	XXXX	cXXX@XXXX.com
2	29486	KXXXXXXX	Abercrombie	XXXX	kXXX@XXXX.com
3	29489	FXXXXXXX	Adams	XXXX	fXXX@XXXX.com

Unmask Data for a User

- Granted UNMASK permission to the test user:

```
1 GRANT UNMASK ON dbo.Customers TO [User2-53407970@LODSPRODMCA.onmicrosoft.com];
```

Messages Copy messages

2:33:05 AM Started executing on line 1

2:33:06 AM Query execution time: 00:00:00.345 | Time to process and render results: 00:00:01.500 | Total duration: 00:00:01.845

```
1 SELECT * FROM dbo.Customers;
```

	123 CustomerID	ABC FirstName	ABC LastName	ABC Phone	ABC Email
1	29485	Catherine	Abel	555-555-5555	catherine0@adventure-works.com
2	29486	Kim	Abercrombie	444-444-4444	kim2@adventure-works.com
3	29489	Frances	Adams	333-333-3333	frances0@adventure-works.com

Apply Row-Level Security

- Created a table for sales data:

secure-data_wh

Schemas

dbo

Tables

Customi

Sales

Views

Functions

Stored Pr...

INFORMATIO...

```

1 CREATE TABLE dbo.Sales
2 (
3     OrderID INT,
4     SalesRep VARCHAR(60),
5     Product VARCHAR(10),
6     Quantity INT
7 );
8
9 --Populate the table with 6 rows of data, showing 3 orders for each test user.
10 INSERT dbo.Sales (OrderID, SalesRep, Product, Quantity) VALUES
11 (1, 'User1-53407970@LODSPRODMCA.onmicrosoft.com', 'Valve', 5),
12 (2, 'User1-53407970@LODSPRODMCA.onmicrosoft.com', 'Wheel', 2),
13 (3, 'User1-53407970@LODSPRODMCA.onmicrosoft.com', 'Valve', 4),
14 (4, 'User2-53407970@LODSPRODMCA.onmicrosoft.com', 'Bracket', 2),
15 (5, 'User2-53407970@LODSPRODMCA.onmicrosoft.com', 'Wheel', 5),
16 (6, 'User2-53407970@LODSPRODMCA.onmicrosoft.com', 'Seat', 5);
17
18 SELECT * FROM dbo.Sales;
```

	123 OrderID	ABC SalesRep	ABC Product	123 Quantity
1	1	User1-53407970@LODSPRODMCA.onmicrosoft.com	Valve	5
2	2	User1-53407970@LODSPRODMCA.onmicrosoft.com	Wheel	2
3	3	User1-53407970@LODSPRODMCA.onmicrosoft.com	Valve	4
4	4	User2-53407970@LODSPRODMCA.onmicrosoft.com	Bracket	2
5	5	User2-53407970@LODSPRODMCA.onmicrosoft.com	Wheel	5
6	6	User2-53407970@LODSPRODMCA.onmicrosoft.com	Seat	5

- Created a security predicate and policy:

```

1  --Create a separate schema to hold the row-level security objects (the predicate function and the security policy)
2  CREATE SCHEMA rls;
3  GO
4
5  /*Create the security predicate defined as an inline table-valued function.
6  A predicate evaluates to true (1) or false (0). This security predicate returns 1,
7  meaning a row is accessible, when a row in the SalesRep column is the same as the user
8  executing the query.*/
9  --Create a function to evaluate who is querying the table
10 CREATE FUNCTION rls.fn_securitypredicate(@SalesRep AS VARCHAR(60))
11 | RETURNS TABLE
12 WITH SCHEMABINDING
13 AS
14 | RETURN SELECT 1 AS fn_securitypredicate_result
15 WHERE @SalesRep = USER_NAME();
16 GO
17 /*Create a security policy to invoke and enforce the function each time a query is run on the Sales table.
18 The security policy has a filter predicate that silently filters the rows available to
19 read operations (SELECT, UPDATE, and DELETE). */
20 CREATE SECURITY POLICY SalesFilter
21 ADD FILTER PREDICATE rls.fn_securitypredicate(SalesRep)
22 ON dbo.Sales
23 WITH (STATE = ON);
24 GO

```

▼ secure-data_wh

▼ Schemas

▼ dbo

▼ Tables

> Customers

> Sales

> Views

> Functions

> Stored Procedures

> INFORMATION_SCHEMA

> queryinsights

▼ rls

> Tables

> Views

▼ Functions

fx fn_securitypredicate

Implement Column-Level Security

- Created a table for orders:

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'secure-data_wh' database is expanded, showing the 'dbo' schema and the 'Orders' table. The main pane displays the following SQL script:

```
1 CREATE TABLE dbo.Orders
2 (
3     OrderID INT,
4     CustomerID INT,
5     CreditCard VARCHAR(20)
6 );
7 INSERT dbo.Orders (OrderID, CustomerID, CreditCard) VALUES
8 (1234, 5678, '1111111111111111'),
9 (2341, 6785, '2222222222222222'),
10 (3412, 7856, '3333333333333333');
11 SELECT * FROM dbo.Orders;
```

Below the script, the 'Results' tab shows the output of the SELECT statement:

	OrderID	CustomerID	CreditCard
1	1234	5678	1111111111111111
2	2341	6785	2222222222222222
3	3412	7856	3333333333333333

- Denied access to the CreditCard column:

```
1 DENY SELECT ON dbo.Orders (CreditCard) TO [User2-53407970@L0DSPRODMCA.onmicrosoft.com];
```

```
1 SELECT * FROM dbo.Orders;
```

Messages Copy messages

3:12:07 AM Started executing on line 1
Msg 230, Level 14, State 1, Line 1
The SELECT permission was denied on the column 'CreditCard' of the object 'Orders', database 'secure-data_wh', schema 'dbo'.

3:12:08 AM Query execution time: 00:00:00.005 | Time to process and render results: 00:00:01.077 | Total duration: 00:00:01.082

```
1 -- SELECT * FROM dbo.Orders;
2
3 SELECT OrderID, CustomerID from dbo.Orders
```

Messages Results Search

	OrderID	CustomerID
1	1234	5678
2	2341	6785
3	3412	7856

Configure SQL Granular Permissions

- Created a stored procedure and a table:

```
1 CREATE PROCEDURE dbo.sp_PrintMessage
2 AS
3 PRINT 'Hello World.';
4 GO
5 CREATE TABLE dbo.Parts
6 (
7     PartID INT,
8     PartName VARCHAR(25)
9 );
10
11 INSERT dbo.Parts (PartID, PartName) VALUES
12 (1234, 'Wheel'),
13 (5678, 'Seat');
14 GO
15
16 /*Execute the stored procedure and select from the table and note the results you get
17 as a member of the Workspace Admin role. Look for output from the stored procedure on
18 the 'Messages' tab.*/
19 EXEC dbo.sp_PrintMessage;
20 GO
21 SELECT * FROM dbo.Parts
```

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'secure-data_wh' database is expanded, showing the 'dbo' schema and the 'Parts' table. The 'Parts' table is selected, and the 'Messages' tab is active. The 'Results' pane shows the following data:

PartID	PartName
1234	Wheel
5678	Seat

- Denied SELECT permissions and granted EXECUTE on the procedure:

```
1 DENY SELECT on dbo.Parts to [User2-53407970@LODSPRODMCA.onmicrosoft.com];
2
3 GRANT EXECUTE on dbo.sp_PrintMessage to [User2-53407970@LODSPRODMCA.onmicrosoft.com];
```

```
1 EXEC dbo.sp_PrintMessage;
2 GO
3
4 SELECT * FROM dbo.Parts;
```

The screenshot shows the SQL Server Enterprise Manager interface. The 'Messages' tab is active, and the 'Results' pane shows the following data:

PartID	PartName
1234	Wheel
5678	Seat

The 'Messages' pane shows the following messages:

- 3:25:34 AM Started executing on line 1
- 3:25:35 AM Hello World.
- 3:25:35 AM The SELECT permission or external policy action 'Microsoft.Sql/Servers/Databases/Schemas/Tables/Rows/Select' was denied on the object 'Parts', database 'secure-data_wh', schema 'dbo'.
- 3:25:35 AM Query execution time: 00:00:00.011 | Time to process and render results: 00:00:01.222 | Total duration: 00:00:01.233

Results

- ✓ A workspace and data warehouse were successfully created in Microsoft Fabric.
- ✓ Dynamic data masking was applied to the Customers table, ensuring sensitive data was masked for users without permissions.
- ✓ Row-level security was implemented, restricting access to sales data based on user identity.
- ✓ Column-level security was enforced, preventing access to the CreditCard column for specific users.
- ✓ SQL granular permissions were configured, allowing for precise control over user access to stored procedures and tables.

Conclusion

This project provided a comprehensive introduction to securing data in a Microsoft Fabric data warehouse. Key security features such as dynamic data masking, row-level security, column-level security, and granular permissions were effectively implemented. These measures ensured that sensitive data is protected while allowing authorized users to access necessary information, demonstrating the robust security capabilities of Microsoft Fabric.

Resources

GitHub: <https://github.com/ThatoMTNG/Microsoft-Fabric-Analytics-Engineer-DP-600->

Mentions

Project Author: Thato Metsing (<https://www.linkedin.com/in/thatometsing/>)

Project Mentor: Maureen Direro (<https://www.linkedin.com/in/maureen-direro-46a6b220/>)