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.

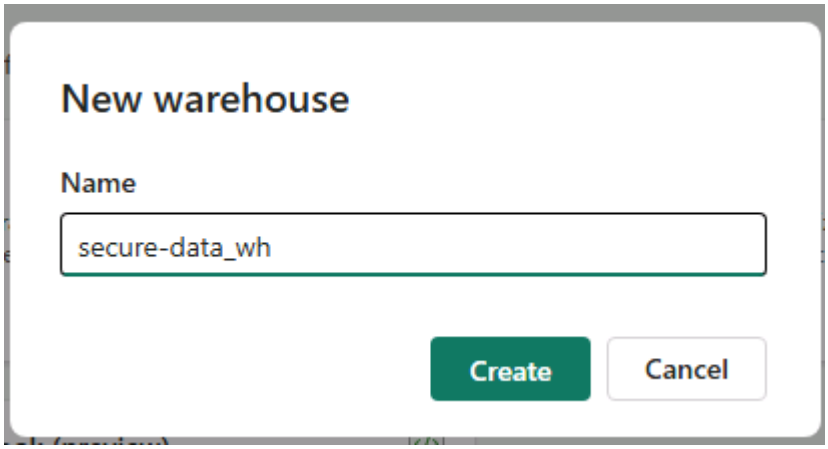
Experience

Create a Workspace

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

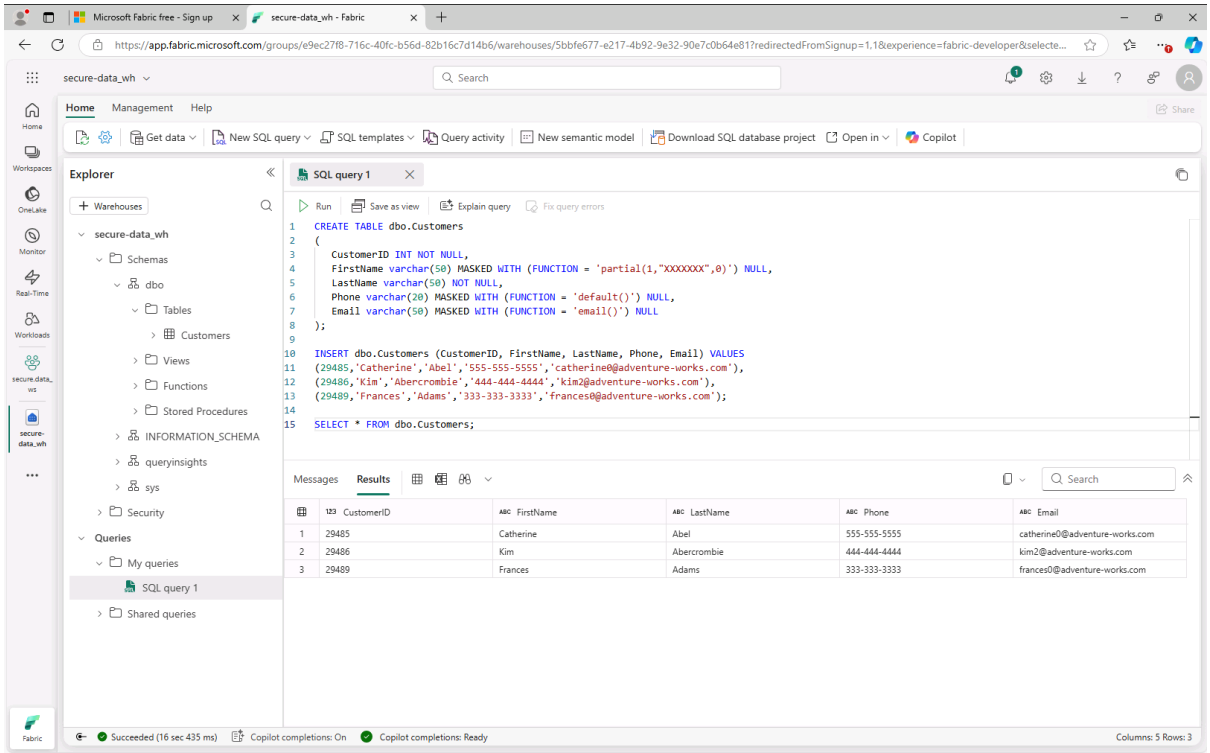
Create a Data Warehouse

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

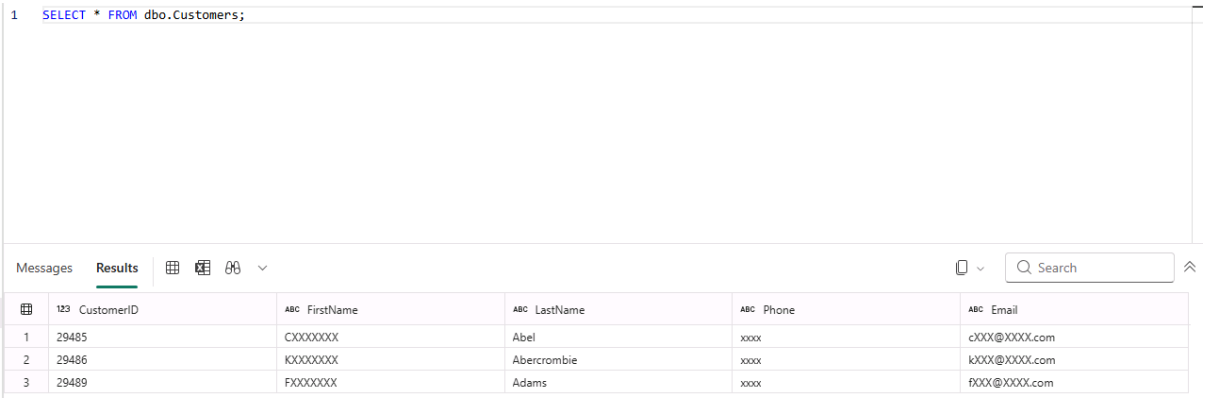


Apply Dynamic Data Masking

- Created a table with dynamic data masking:



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



Unmask Data for a User

- Granted UNMASK permission to the test user:

```sql
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

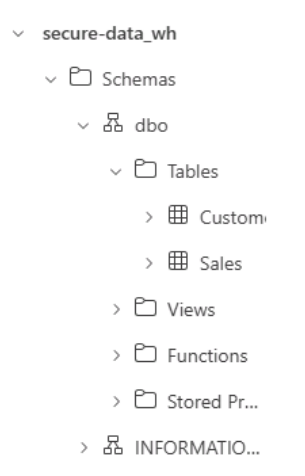
```sql
SELECT * FROM dbo.Customers;
```

Messages Results Search

	CustomerID	FirstName	LastName	Phone	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:

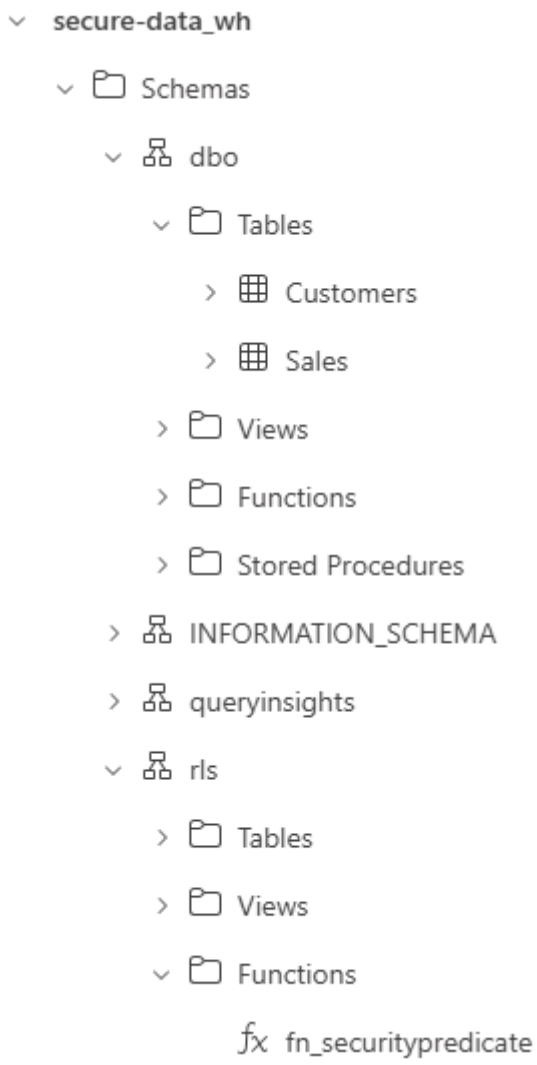



```sql
CREATE TABLE dbo.Sales
(
    OrderID INT,
    SalesRep VARCHAR(60),
    Product VARCHAR(10),
    Quantity INT
);

--Populate the table with 6 rows of data, showing 3 orders for each test user.
INSERT dbo.Sales (OrderID, SalesRep, Product, Quantity) VALUES
(1, 'User1-53407970@LODSPRODMCA.onmicrosoft.com', 'Valve', 5),
(2, 'User1-53407970@LODSPRODMCA.onmicrosoft.com', 'Wheel', 2),
(3, 'User1-53407970@LODSPRODMCA.onmicrosoft.com', 'Valve', 4),
(4, 'User2-53407970@LODSPRODMCA.onmicrosoft.com', 'Bracket', 2),
(5, 'User2-53407970@LODSPRODMCA.onmicrosoft.com', 'Wheel', 5),
(6, 'User2-53407970@LODSPRODMCA.onmicrosoft.com', 'Seat', 5);

SELECT * FROM dbo.Sales;
```

	OrderID	SalesRep	Product	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

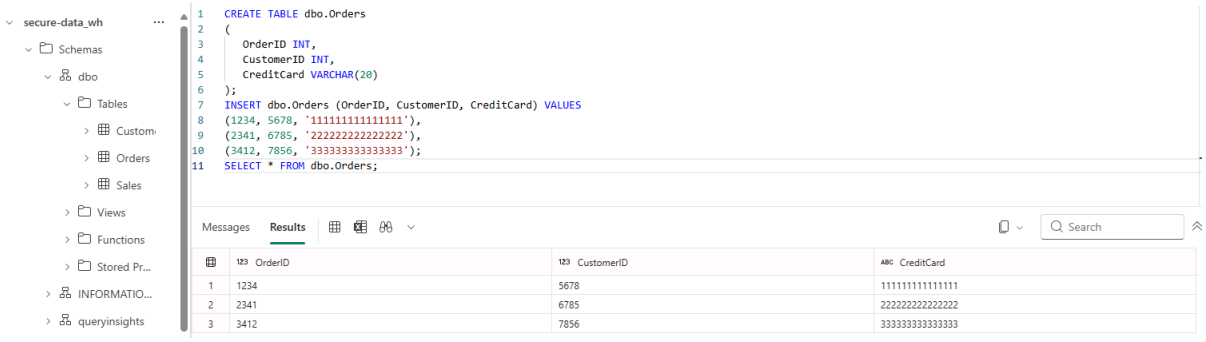
- Created a security predicate and policy:

```sql
--Create a separate schema to hold the row-level security objects (the predicate function and the security policy)
CREATE SCHEMA rls;
GO

/*Create the security predicate defined as an inline table-valued function.
A predicate evaluates to true (1) or false (0). This security predicate returns 1,
meaning a row is accessible, when a row in the SalesRep column is the same as the user
executing the query.*/
--Create a function to evaluate who is querying the table
CREATE FUNCTION rls.fn_securitypredicate(@SalesRep AS VARCHAR(60))
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN SELECT 1 AS fn_securitypredicate_result
WHERE @SalesRep = USER_NAME();
GO
/*Create a security policy to invoke and enforce the function each time a query is run on the Sales table.
The security policy has a filter predicate that silently filters the rows available to
read operations (SELECT, UPDATE, and DELETE). */
CREATE SECURITY POLICY SalesFilter
ADD FILTER PREDICATE rls.fn_securitypredicate(SalesRep)
ON dbo.Sales
WITH (STATE = ON);
GO
```



Implement Column-Level Security

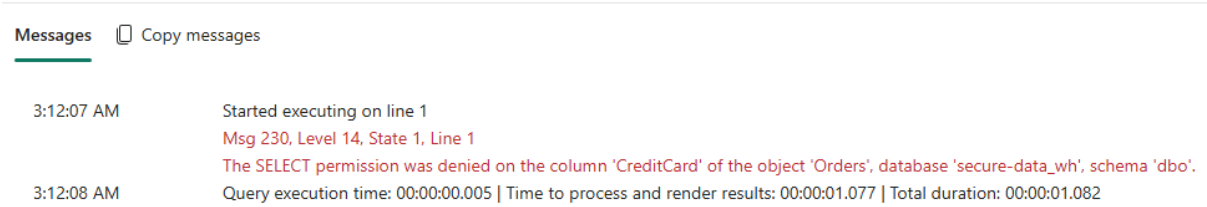
- Created a table for orders:

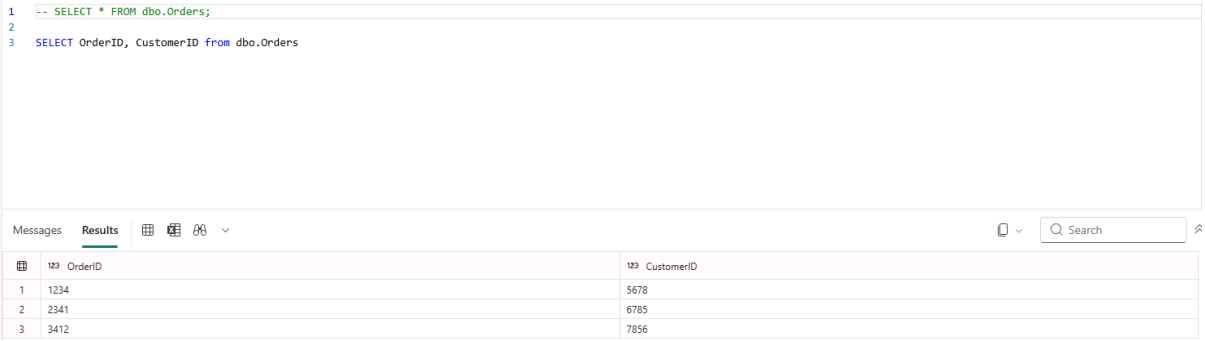


- Denied access to the CreditCard column:

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

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

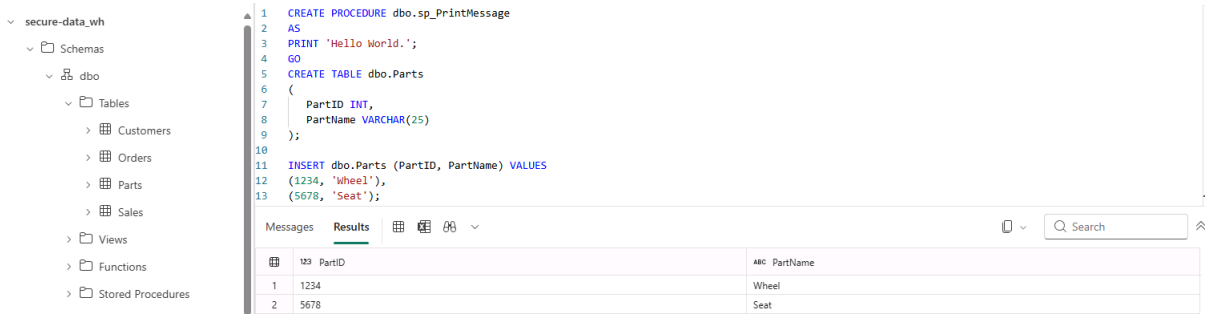




Configure SQL Granular Permissions

- Created a stored procedure and a table:

```sql
CREATE PROCEDURE dbo.sp_PrintMessage
AS
PRINT 'Hello World.';
GO
CREATE TABLE dbo.Parts
(
    PartID INT,
    PartName VARCHAR(25)
);

INSERT dbo.Parts (PartID, PartName) VALUES
(1234, 'Wheel'),
(5678, 'Seat');
GO

/*Execute the stored procedure and select from the table and note the results you get
as a member of the Workspace Admin role. Look for output from the stored procedure on
the 'Messages' tab.*/
EXEC dbo.sp_PrintMessage;
GO
SELECT * FROM dbo.Parts
```



- Denied SELECT permissions and granted EXECUTE on the procedure:

```sql
DENY SELECT on dbo.Parts to [User2-53407970@LODSPRODMCA.onmicrosoft.com];

GRANT EXECUTE on dbo.sp_PrintMessage to [User2-53407970@LODSPRODMCA.onmicrosoft.com];
```

```sql
EXEC dbo.sp_PrintMessage;
GO

SELECT * FROM dbo.Parts;
```

Messages | Copy messages

```
3:25:34 AM    Started executing on line 1
              Hello World.
              Msg 373, Level 14, State 5, Line 4
              The SELECT permission or external policy action 'Microsoft.Sql/Sqlservers/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/)