ProcureSQL

Advanced Data Protection with Azure SQL Database and SQL Server

John Sterrett
CEO of Procure SQL

About John Sterrett

ProcureSQL

Microsoft
CERTIFIED TRAINER
MCT

john@procuresql.com

procuresql.com

linkedin.com/in/johnsterrett

Goal of Today's Session

Today's Goals

Learn how to secure your data with Auditing, Row-Level Security, and Always Encrypted

Gain insight into pros and cons to utilizing advanced data protection features.

Leverage built in cloud features provided by Azure SQL for advanced data security.

Azure SQL Built In Protection Features

- Azure SQL Firewall & Private Link
- Advanced Threat Protection
- Transparent Data Encryption
- Data Discovery and Classification
- Encryption in-transit
- Vulnerability Assessments

Azure SQL Server – Vulnerability Assessment

Home > Microsoft Defender for Cloud | Inventory >

Resource health

SQL server

5
Active recommendations

0
Active alerts

Resource information

Subscription
MS Silver

Resource Group

Environment
Azure

Location
southcentralus

Status
Ready

Security value

Microsoft Defender for Azure SQL database servers
On

Recommendations | Alerts

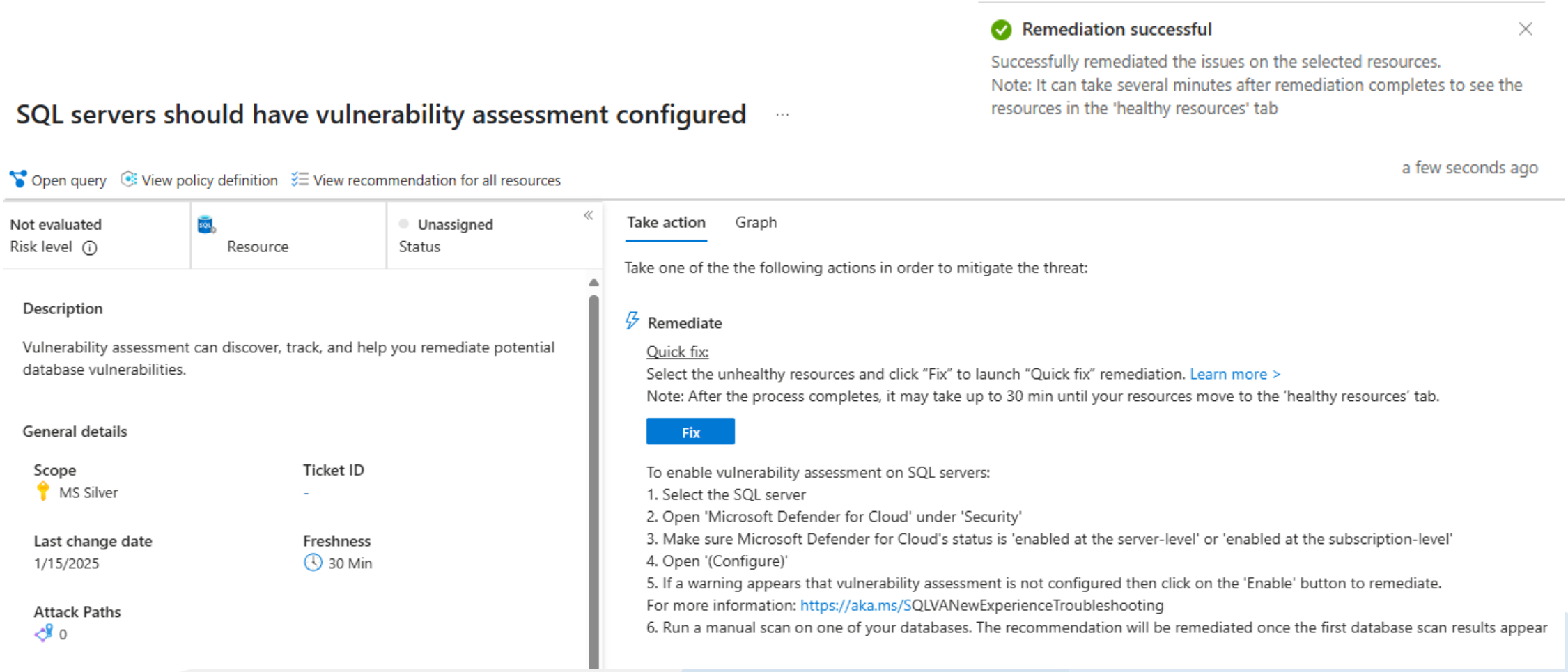
Search

Status == **Unhealthy**

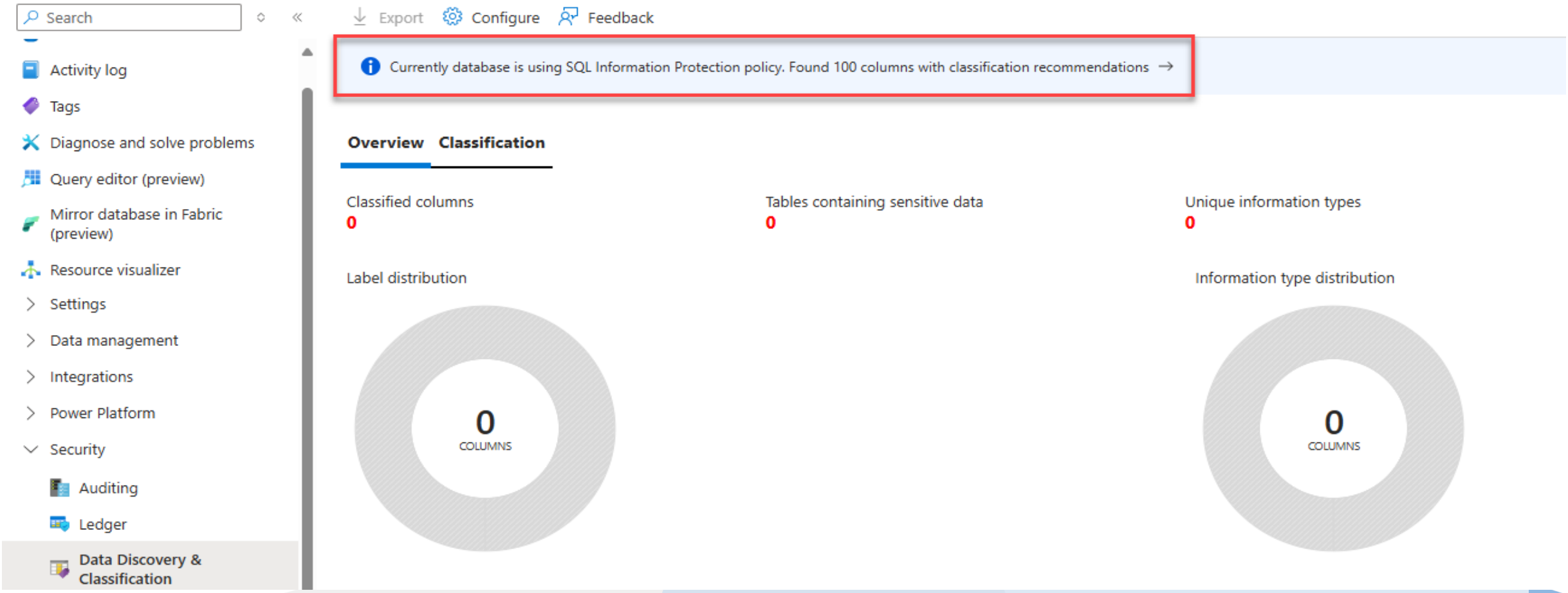
Severity == **All**

Severity	Description	Status
High	SQL servers should have an Azure Active Directory administrator provisioned	Unhealthy
High	SQL databases should have vulnerability findings resolved	Unhealthy
Medium	Azure SQL Database should have Azure Active Directory Only Authentication enabled Preview	Unhealthy
Medium	Public network access on Azure SQL Database should be disabled Preview	Unhealthy
Low	Auditing on SQL server should be enabled	Unhealthy

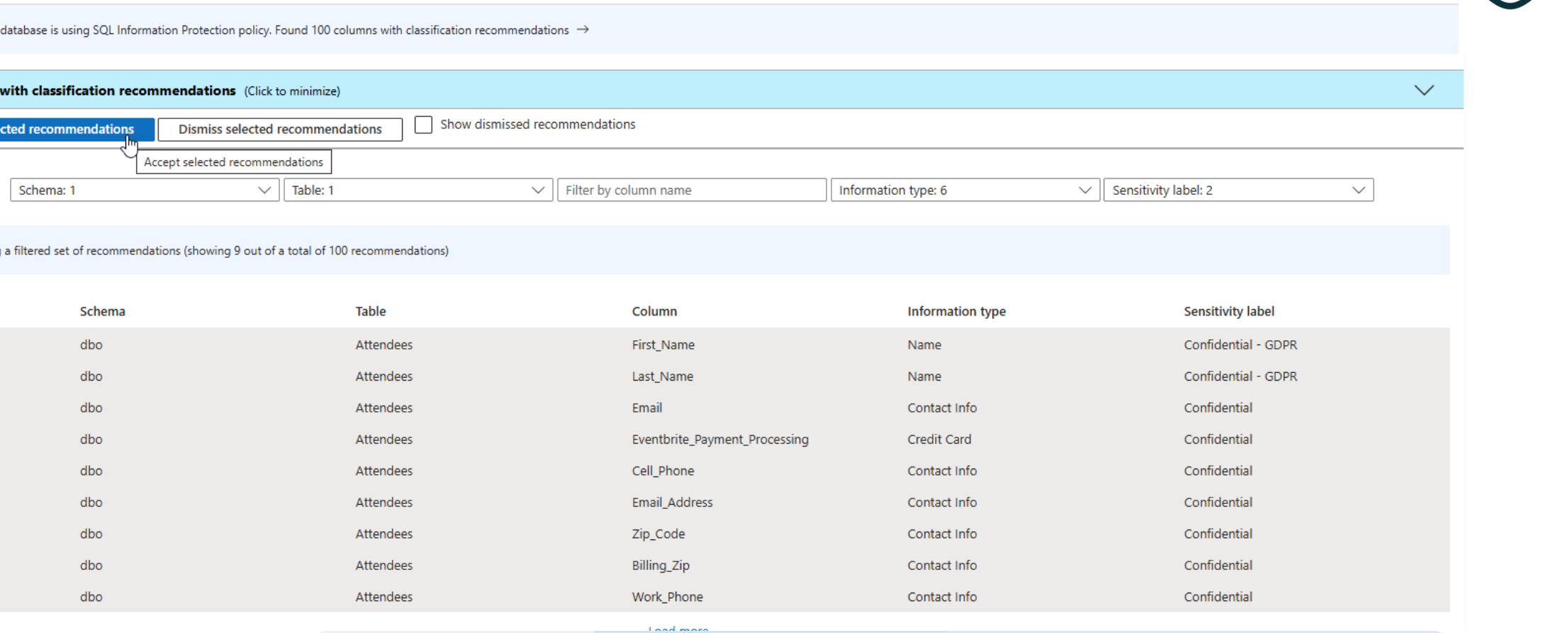
Details and Remediate Steps



Discover and Classify Sensitive Data



Adding Classified Columns



Data Classification after Saving Recommendations

Currently database is using SQL Information Protection policy. Found 100 columns with classification recommendations →

Overview **Classification**

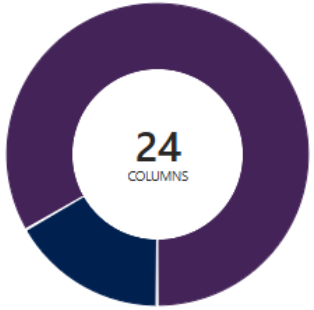
Learn more - Getting Started Guide

Classified columns
24

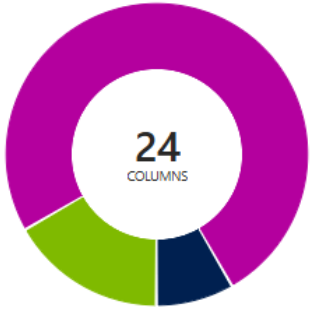
Tables containing sensitive data
2

Unique information types
3

Label distribution



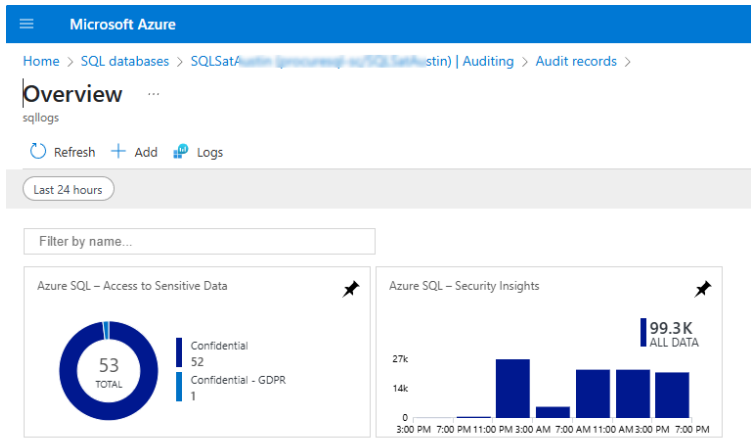
Information type distribution

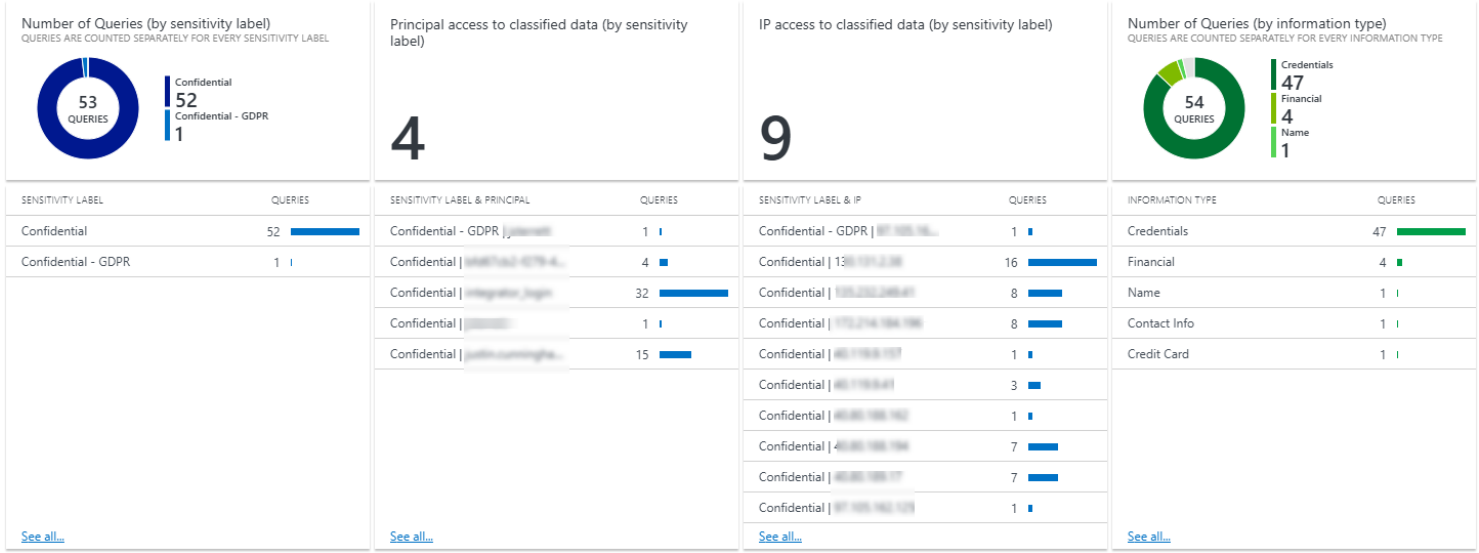


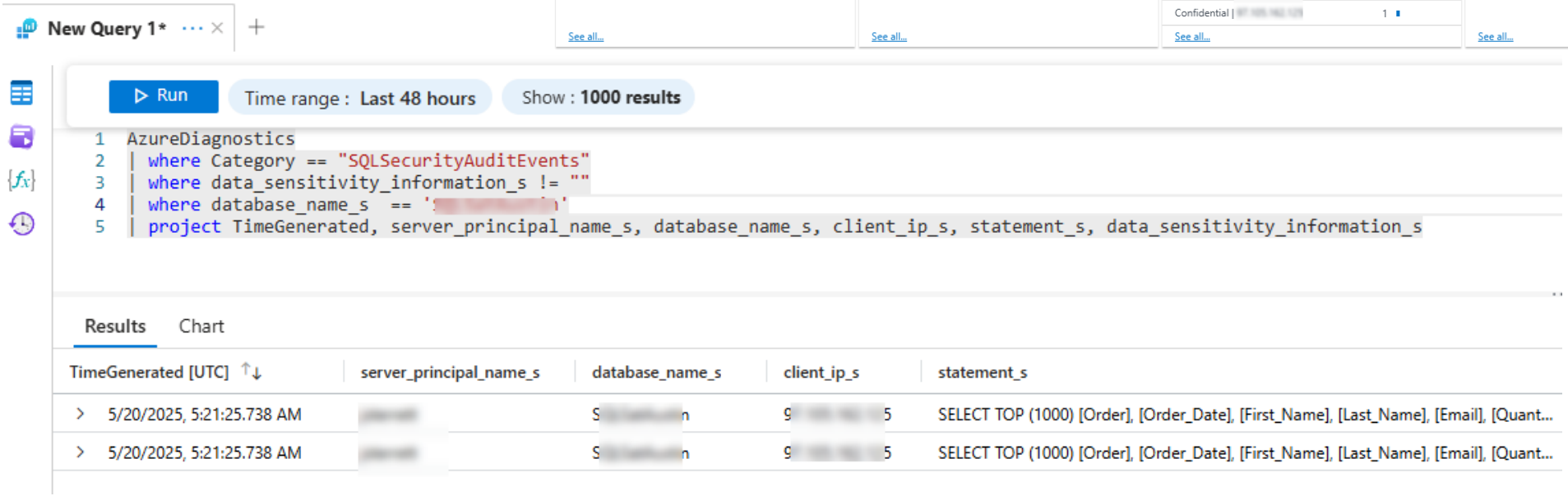
Schema: 1 | Table: 2 | Filter by column name | Information type: 3 | Sensitivity label: 2

Schema	Table	Column	Information type	Sensitivity label
dbo				
	Attendees	First_Name	Name	Confidential - GDPR
	Attendees	Last_Name	Name	Confidential - GDPR
	Attendees	Email	Contact Info	Confidential
	Attendees	Eventbrite_Payment_Processing	Credit Card	Confidential
	Attendees	Cell_Phone	Contact Info	Confidential

Data Sensitive Metrics Found in our SQL Audit







ProcureSQL

ProcureSQL

SQL Audit

Identify Who, What, When, Where behind data access, security and schema changes

What do these compliances have in common?

- ISO 27001
- GDPR
- PCI DSS (Payment Card Industry Data Security Standard)
- HIPAA (Health Insurance Portability and Accountability Act)
- SOX (Sarbanes-Oxley Act)

SQL Audit Enables You To….

Track login attempts, failed logins, which can indicate attempts of malicious attempts

Changes to database schema, permissions, and security configurations

Record Access to both data and object changes in real-time

SQL Audit Decisions

- What do we Audit? Were does
- Audit Logging Go?
- Where do we store audit data?
- What should happen when audit logging fails?
- How much log data do we keep?
- How do we secure the Audit Logs?
- How do we Organize Audits?

What to Audit for HIPAA, PCI & GDPR?

Compliance	Common SQL Audit Actions
HIPAA	SELECT, INSERT, UPDATE, DELETE (data access/modification) on PHI tables; login success/failure; permission changes;
PCI DSS	User account creation/modification/deletion; login attempts; permission changes; data access/modification
GDPR	Logon activity; unauthorized access attempts; data access and processing

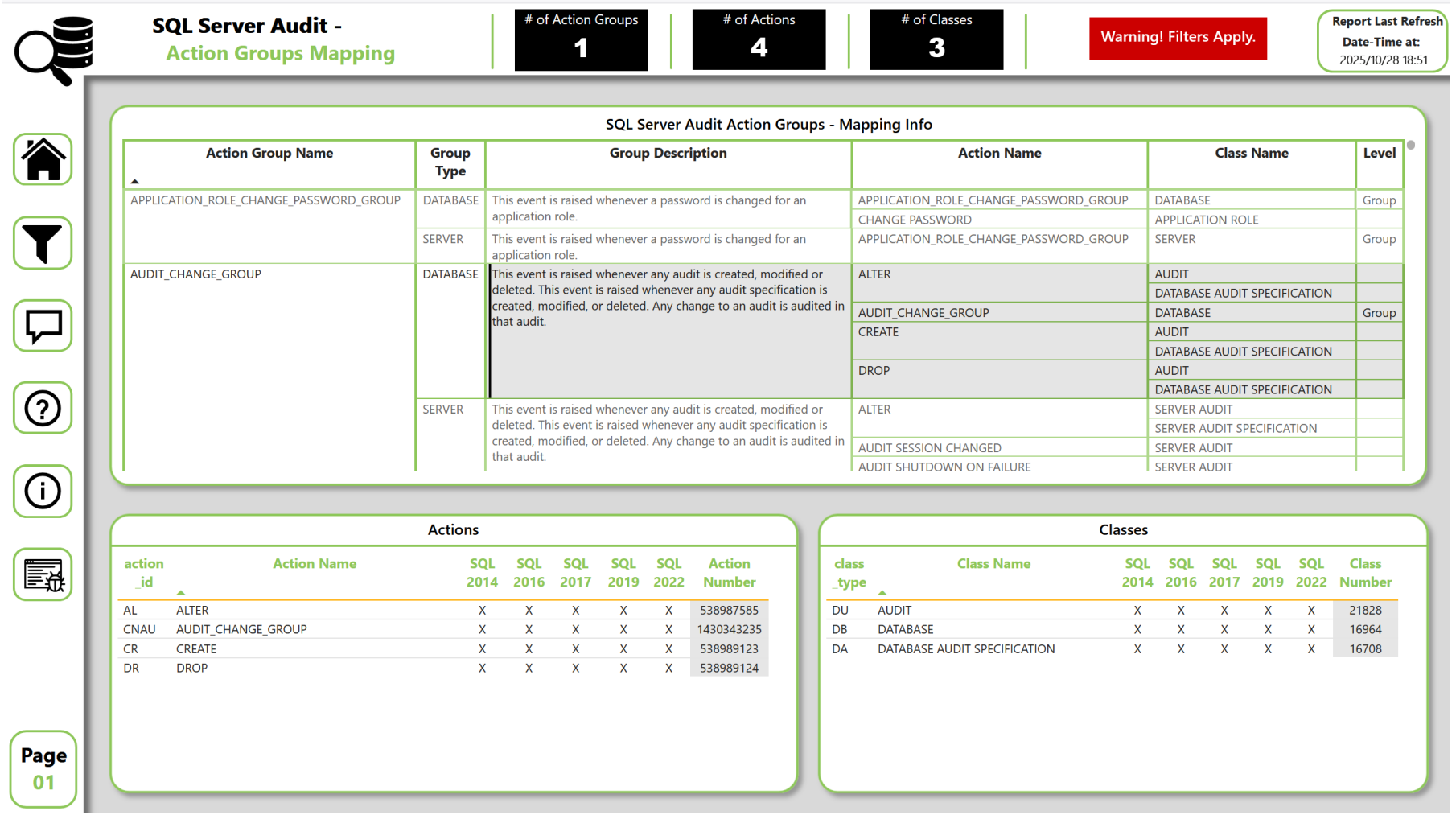
Audit Action Groups for HIPAA, GDPR, PCI

Audit Action Group	Description
SUCCESSFUL_LOGIN_GROUP	Captures all successful login attempts. Important for tracking authorized access (HIPAA, GDPR, PCI).
FAILED_LOGIN_GROUP	Captures failed login attempts, helping detect unauthorized access or brute force attacks (HIPAA, GDPR, PCI).
SERVER_PRINCIPAL_CHANGE_GROUP	Tracks creation, alteration, or deletion of server-level principals (logins, users). Critical for permission management (HIPAA, PCI).
SERVER_ROLE_MEMBER_CHANGE_GROUP	Monitors changes to server role memberships, ensuring role-based access control integrity (HIPAA, PCI).
AUDIT_CHANGE_GROUP	Audits creation, modification, or deletion of audit objects and audit specifications, ensuring audit trail integrity (HIPAA, PCI, GDPR).
APPLICATION_ROLE_CHANGE_PASSWORD_GROUP	Tracks changes to application role passwords, securing application-level access (HIPAA, PCI).
LOGIN_CHANGE_PASSWORD_GROUP	Captures password changes for logins, supporting credential management policies (HIPAA, PCI).
SCHEMA_OBJECT_CHANGE_GROUP	Monitors DDL changes (CREATE, ALTER, DROP) on schema objects at the server level, important for tracking structural changes (HIPAA, GDPR).
DATABASE_OBJECT_PERMISSION_CHANGE_GROUP	Tracks permission changes on database objects, ensuring proper access control (HIPAA, PCI, GDPR).
DATABASE_PRINCIPAL_CHANGE_GROUP	Monitors changes to database principals (users, roles), supporting identity management (HIPAA, PCI).

These server-level action groups are added to server audit specifications, which audit events across the entire SQL Server instance.

For detailed data access auditing (e.g., SELECT, INSERT, UPDATE, DELETE on specific tables), database audit specifications with database-level action groups are used.

SQL Audit Action Groups Cheat Sheet



SQL Server Audit -
Action Groups Mapping

# of Action Groups	# of Actions	# of Classes
1	4	3

Warning! Filters Apply.

Report Last Refresh Date-Time at: 2025/10/28 18:51

SQL Server Audit Action Groups - Mapping Info

Action Group Name	Group Type	Group Description	Action Name	Class Name	Level
APPLICATION_ROLE_CHANGE_PASSWORD_GROUP	DATABASE	This event is raised whenever a password is changed for an application role.	APPLICATION_ROLE_CHANGE_PASSWORD_GROUP	DATABASE	Group
			CHANGE PASSWORD	APPLICATION ROLE	
	SERVER	This event is raised whenever a password is changed for an application role.	APPLICATION_ROLE_CHANGE_PASSWORD_GROUP	SERVER	Group
AUDIT_CHANGE_GROUP	DATABASE	This event is raised whenever any audit is created, modified or deleted. This event is raised whenever any audit specification is created, modified, or deleted. Any change to an audit is audited in that audit.	ALTER	AUDIT	
				DATABASE AUDIT SPECIFICATION	
			AUDIT_CHANGE_GROUP	DATABASE	Group
			CREATE	AUDIT	
				DATABASE AUDIT SPECIFICATION	
			DROP	AUDIT	
				DATABASE AUDIT SPECIFICATION	
	SERVER	This event is raised whenever any audit is created, modified or deleted. This event is raised whenever any audit specification is created, modified, or deleted. Any change to an audit is audited in that audit.	ALTER	SERVER AUDIT	
				SERVER AUDIT SPECIFICATION	
			AUDIT SESSION CHANGED	SERVER AUDIT	
			AUDIT SHUTDOWN ON FAILURE	SERVER AUDIT	

Actions

action_id	Action Name	SQL 2014	SQL 2016	SQL 2017	SQL 2019	SQL 2022	Action Number
AL	ALTER	X	X	X	X	X	538987585
CNAU	AUDIT_CHANGE_GROUP	X	X	X	X	X	1430343235
CR	CREATE	X	X	X	X	X	538989123
DR	DROP	X	X	X	X	X	538989124

Classes

class_type	Class Name	SQL 2014	SQL 2016	SQL 2017	SQL 2019	SQL 2022	Class Number
DU	AUDIT	X	X	X	X	X	21828
DB	DATABASE	X	X	X	X	X	16964
DA	DATABASE AUDIT SPECIFICATION	X	X	X	X	X	16708

Page 01

[Power BI Dashboard Link]

ProcureSQL

SQL Audit – SQL Server vs. Azure SQLDB

SQL Server

- To **create, alter, or drop a server audit**, principals require the **ALTER ANY SERVER AUDIT** or the **CONTROL SERVER** permission.
- Users with the **ALTER ANY SERVER AUDIT** permission can create server audit specifications and bind them to any audit.
- After a server audit specification is created, it can be viewed by principals with the **CONTROL SERVER or ALTER ANY SERVER AUDIT** permissions, the **sysadmin** account, or **principals having explicit access to the audit**.

Azure SQL Database

- Need Contributor role or higher on the database or server resource
  - Permissions to execute 'Microsoft.Sql/servers/extendedAuditingSettings/write'
  - Permission to execute 'Microsoft.Sql/servers/databases/extendedAuditingSettings/write'
- The following audit policies are included by default
  - BATCH_COMPLETED_GROUP
    SUCCESSFUL_DATABASE_AUTHENTICATION_GROUP
    FAILED_DATABASE_AUTHENTICATION_GROUP
- Additional changes can be made via API calls.
- Enabling auditing on the database in addition to enabling auditing on the server doesn't override or change any of the settings of the server auditing

SQL Audit Targets - Storage

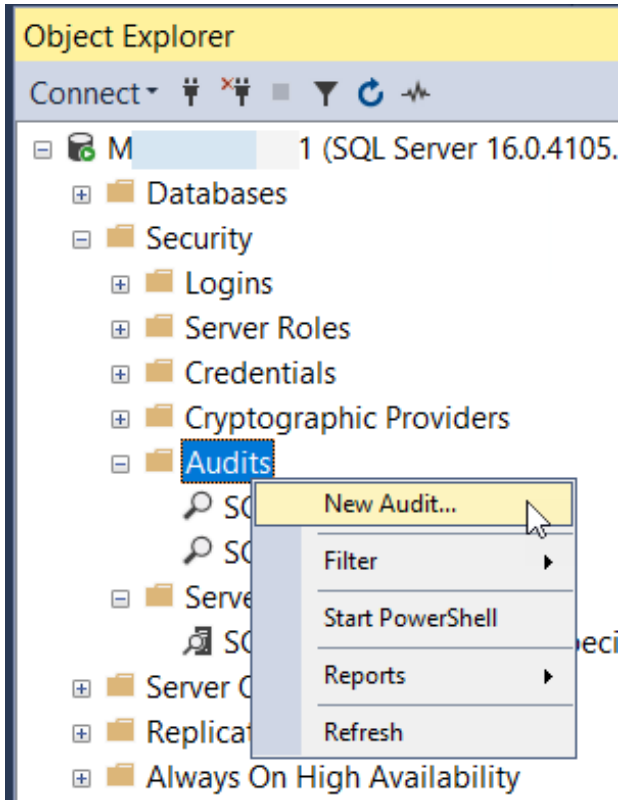
Targets include

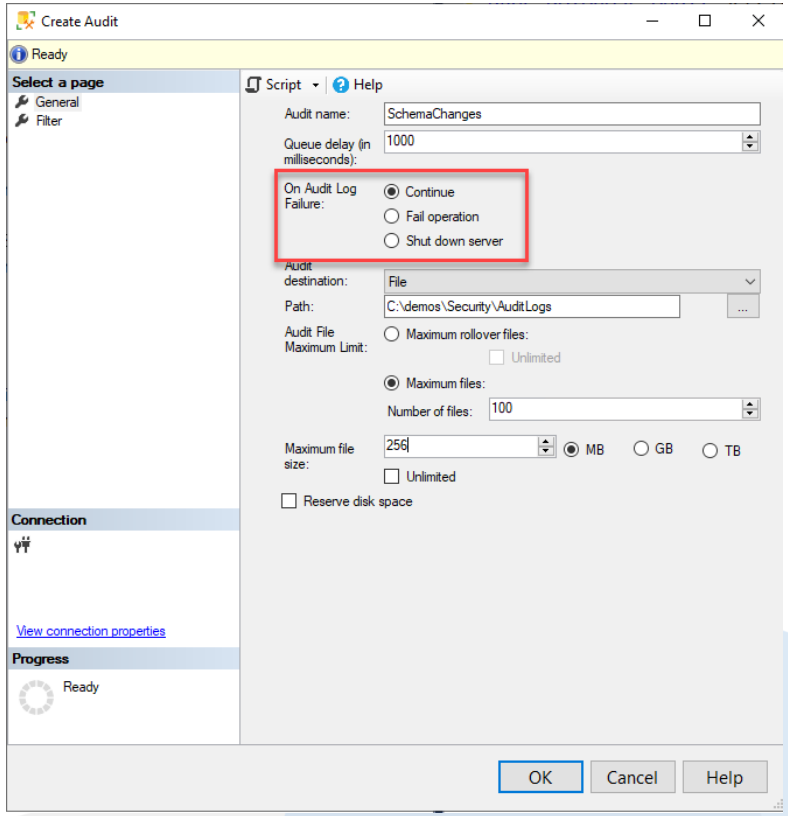
- Azure - Blob storage, Log Analytics, Event Hub
- SQL Server – Files, Event Logs

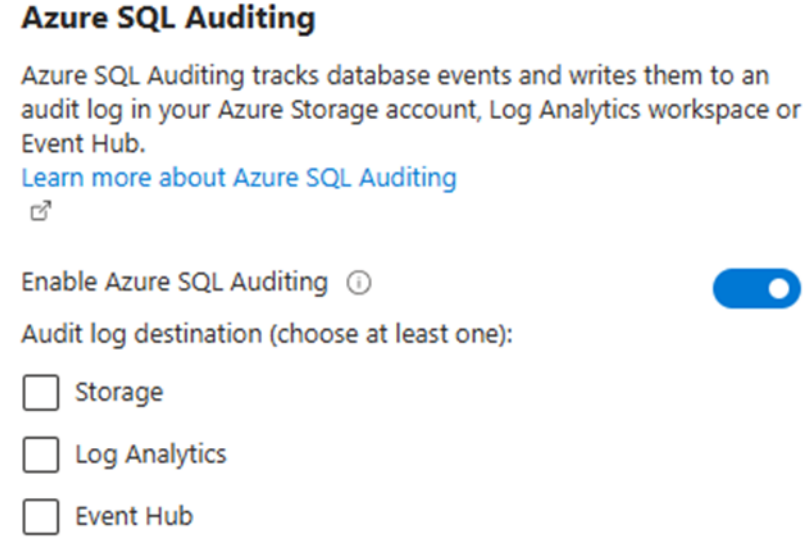
Use file targets with appropriate **size limits (MAXSIZE)** and rollover files **(MAX_ROLLOVER_FILES)** to **prevent disk space issues.**

Reserve disk space upfront **(RESERVE_DISK_SPACE = ON)** to **avoid audit failures due to insufficient space**.

Creating a SQL Audit

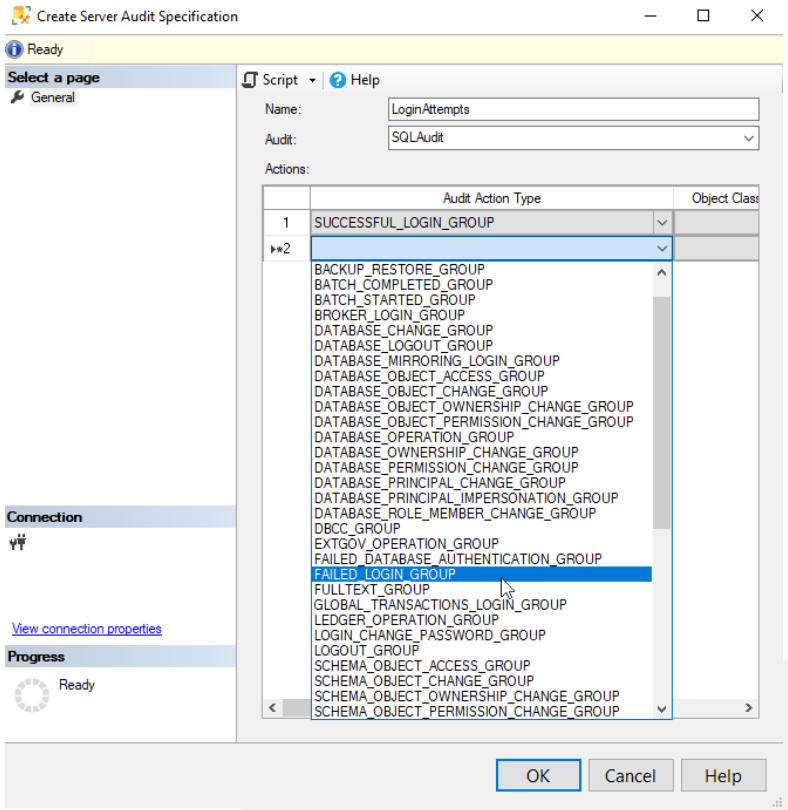


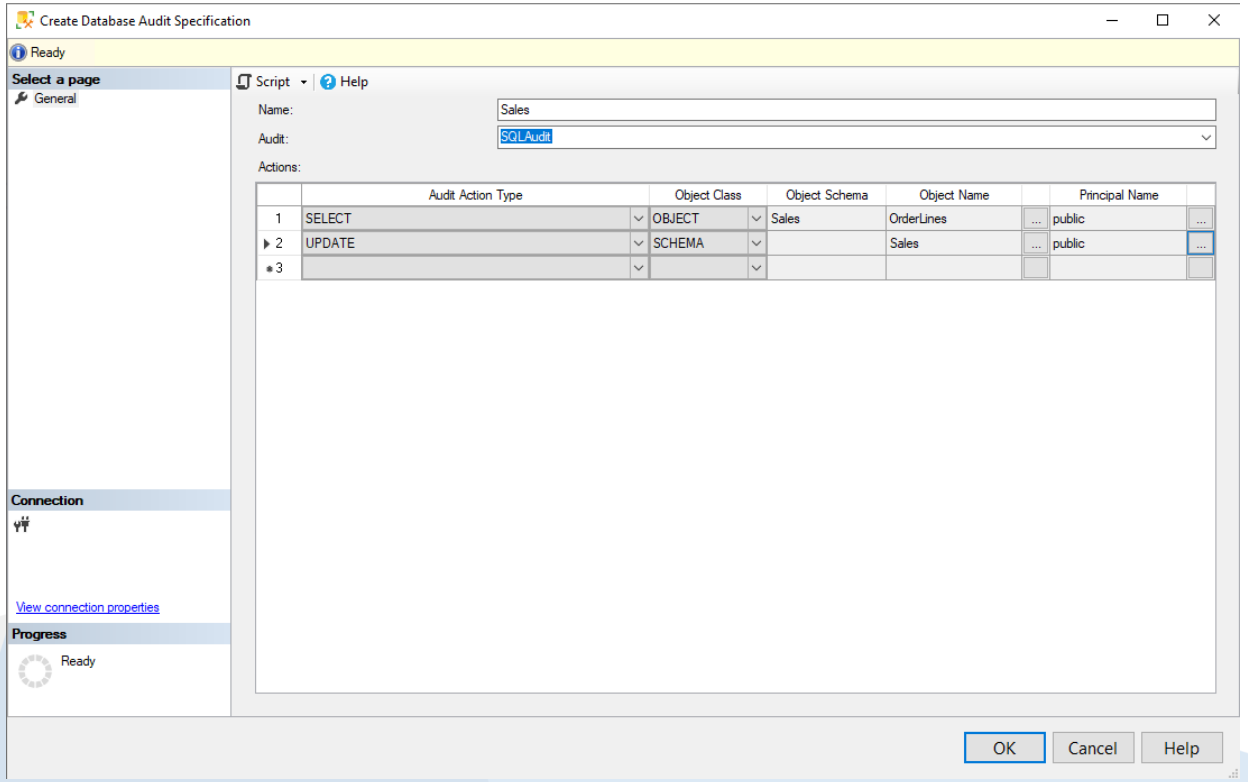




How to Audit events?

Server Audit Specifications and **Database Audit Specifications** are used to specify what events we will audit.





T-SQL Basic Example

```
CREATE SERVER AUDIT
Audit_Compliance TO FILE (
FILEPATH = 'C:\AuditLogs\')
WITH (ON_FAILURE =
CONTINUE);
GO

ALTER SERVER AUDIT
Audit_Compliance WITH (STATE
= ON);
GO
```

```
CREATE SERVER AUDIT SPECIFICATION AuditSpec_Compliance
FOR SERVER AUDIT Audit_Compliance
ADD (SUCCESSFUL_LOGIN_GROUP),
ADD (FAILED_LOGIN_GROUP),
ADD (SERVER_PRINCIPAL_CHANGE_GROUP),
ADD (SERVER_ROLE_MEMBER_CHANGE_GROUP),
ADD (AUDIT_CHANGE_GROUP),
ADD (LOGIN_CHANGE_PASSWORD_GROUP),
ADD (APPLICATION_ROLE_CHANGE_PASSWORD_GROUP);
GO
ALTER SERVER AUDIT SPECIFICATION AuditSpec_Compliance
WITH (STATE = ON);
GO

CREATE DATABASE AUDIT [CustomerDataAudit]
FOR SERVER AUDIT [SQLAudit]
ADD (SELECT ON OBJECT::[dbo].[Customers] BY [public]),
ADD (UPDATE ON OBJECT::[dbo].[Customers] BY [public]),
ADD (INSERT ON OBJECT::[dbo].[Customers] BY [public]),
ADD (DELETE ON OBJECT::[dbo].[Customers] BY [public])
```

Viewing Audit with Log File Viewer

Log File Viewer -

Select logs

- [x] Audit Collection
  - [ ] .
  - [x] SQLAudit
  - [ ] SchemaChanges

Load Log | Export | Refresh | Filter ... | Search ... | Stop | Help

Log file summary: No filter applied

Action ID	Class Type	Database Principal Name	Session Server Principal Name	Object Name	Schema Name	Database Name	Statement
SELECT	TABLE	dbo	Pl[illegible]lm	Orders	Sales	WideWorldImporters	SELECT TOP (1000) [OrderID]
SELECT	TABLE	dbo	Pl[illegible]lm	Orders	Sales	WideWorldImporters	SELECT TOP (1000) [OrderID]
SELECT	TABLE	dbo	Pl[illegible]lm	Orders	Sales	WideWorldImporters	SELECT TOP (1000) [OrderID]
SELECT	TABLE	dbo	Pl[illegible]lm	Orders	Sales	WideWorldImporters	SELECT TOP (1000) [OrderID]
SELECT	TABLE	dbo	Pl[illegible]lm	Orders	Sales	WideWorldImporters	SELECT TOP (1000) [OrderID]

```sql
;with cte as (
select *
FROM sys.fn_get_audit_file ('C:\demos\Security\AuditLogs\SQLAudit*'
, DEFAULT,  DEFAULT)  )
SELECT event_time, [session_server_principal_name],
client_ip, application_name as AppName,
database_name, [schema_name], [object_name], statement
FROM cte
WHERE [database_name] like 'WideWorldImporters'
```

100 %

Results | Messages

	event_time	session_server_principal_name	client_ip	AppName	database_name	schema_name	object_name	statement
1	2025-05-18 20:24:4...	P[illegible]m	local machine	Microsoft SQL Server Management Studio - Query	WideWorldImporters	Sales	Orders	SELECT TOP (1000) [OrderID] ,[CustomerID]
2	2025-05-18 20:25:5...	P[illegible]m	local machine	Microsoft SQL Server Management Studio - Query	WideWorldImporters	Sales	Orders	SELECT TOP (1000) [OrderID] ,[CustomerID]
3	2025-05-18 20:26:2...	P[illegible]m	local machine	Microsoft SQL Server Management Studio - Query	WideWorldImporters	Sales	Orders	SELECT TOP (1000) [OrderID] ,[CustomerID]
4	2025-05-18 20:26:2...	P[illegible]m	local machine	Microsoft SQL Server Management Studio - Query	WideWorldImporters	Sales	Orders	SELECT TOP (1000) [OrderID] ,[CustomerID]
5	2025-05-18 20:26:2...	P[illegible]m	local machine	Microsoft SQL Server Management Studio - Query	WideWorldImporters	Sales	Orders	SELECT TOP (1000) [OrderID] ,[CustomerID]

ProcureSQL

SQL Audit Best Practices

Define	Clear Audit Goals and Scope
Use	Server and Database Audit specifications – Limit Tracked Events
Review	Audit Logs Regularly
Secure	Secure Your Audit Logs
Backup	Your Audit Logs
Audit	Your Audit - Verify intended events are being logged

ProcureSQL

Row-Level Security (RLS)

Securing Your Data at the Row Level

Why Row-Level Security?

Applications need to limit a users access to only certain rows of data in a database. **Security needs to be embedded in the database to work for ALL APPLICATIONS.**

- Control both read and write data at the row level
- No app changes needed, works transparently when queries execute
- Centralized Security Logic within the database
- Apps consume secured data
  - Excel, .NET, **Power BI – Direct Query**, etc.…

RLS Real-World Examples

Health Care (Patient Data Access Controls)

- Nurses can only view rows of their assigned patients
- Doctors access broader data but are blocked from data unless authorized
- Patients can only see their data

Financial Services

- Financial Advisors only see their client’s portfolios
- Auditors access transition history for only the Financial Advisors they audit

E-Commerce / Multi-Tenant

- Vendors view only their sales records and customer orders
- Platform admins access data for vendors assigned to them.

How does RLS Work?

Predicate-based access control added to regular access

Two types of security predicates

Filter predicates – silently filter SELECT, UPDATE and DELETE operations to exclude rows that will not satisfy the predicate

Block predicates – block INSERT, UPDATE, DELETE operations that will not satisfy the predicate

AFTER INSERT and AFTER UPDATE predicates can prevent users from updating rows to values that violate the predicate.

BEFORE UPDATE predicates can prevent users from updating rows that currently violate the predicate.

BEFORE DELETE predicates can block delete operations.

How to Implement RLS

Each row of your table has column that determine which user can access the data

dbo.Customers

CustomerID	FirstName	LastName	SalesRep
1001			John
1002			Kon
1003			John

Create inline table-value function that defines row level access criteria

```
CREATE FUNCTION RLS.CustomerPredicate (@SalesRep AS sysname)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN SELECT 1 AS Access
WHERE @SalesRep = USER_NAME() OR USER_NAME() = 'Manager'
GO
```

Security policy adds security predicates on tables using the function provided

```
CREATE SECURITY POLICY RLS.CustomerPolicy
ADD FILTER PREDICATE RLS.CustomerPredicate(SalesRep) ON Sales.Customer,
ADD BLOCK PREDICATE RLS.CustomerPredicate(SalesRep) ON Sales.Customer
GO
```

Authorization Methods with RLS

Any lookup defined by business rules can be used

Lookup Options

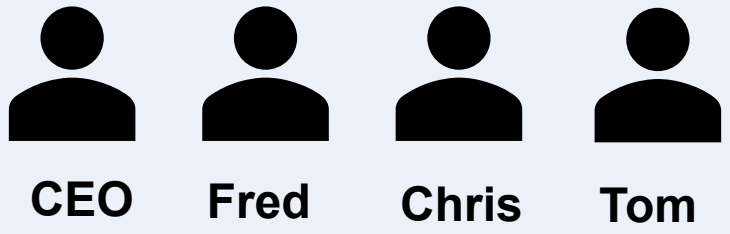
- SESSION_CONTEXT() – Applications
- Local Lookup table
- SQL Roles
- Specific users – admins as an example

RLS Example (Group Activity)

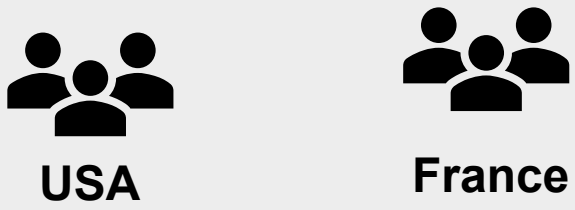
dbo.Sales - Table

User Name	Country	Sales
Fred	USA	10000
Chris	USA	9500
Tom	France	9600
Fred	Spain	9200
Chris	Germany	9000

Database Users



Database Roles



RLS – User lookup Example

```
CREATE FUNCTION
Security.fn_SalesSecurity(@UserName AS sysname)
RETURNS TABLE
WITH SCHEMABINDING
AS

   RETURN SELECT 1 AS fn_SalesSecurity_Result
   -- Logic for filter predicate
   WHERE @UserName = USER_NAME() OR USER_NAME() =
'CEO';
GO

CREATE SECURITY POLICY Security.UserFilter
ADD FILTER PREDICATE
Security.fn_SalesSecurity(UserName)
ON dbo.Sales WITH (STATE = ON);
GO

EXECUTE AS USER = 'CEO';
SELECT * FROM Sales;
REVERT;
GO

EXECUTE AS USER = 'Fred';
SELECT * FROM Sales;
REVERT;
```

Database User

CEO

Database User

Fred

dbo.Sales - Table

User Name	Country	Sales
Fred	USA	10000
Chris	USA	9500
Tom	France	9600
Fred	Spain	9200
Chris	Germany	9000

User Name	Country	Sales
Fred	USA	10000
Fred	Spain	9200

RLS – SQL Roles Example

```
ALTER ROLE [USA] ADD MEMBER [CEO]
ALTER ROLE [FRANCE] ADD MEMBER [CEO]
ALTER ROLE [SPAIN] ADD MEMBER [CEO]

ALTER ROLE [USA] ADD MEMBER [Fred]
ALTER ROLE [USA] ADD MEMBER [Chris]


ALTER ROLE [FRANCE] ADD MEMBER [Tom]
ALTER ROLE [SPAIN] ADD MEMBER [Fred]
ALTER ROLE [GERMANY] ADD MEMBER [Chris]

CREATE FUNCTION Security.fn_SalesSecurity(@RoleName AS
sysname)
RETURNS TABLE
WITH SCHEMABINDING AS
RETURN ( SELECT 1 AS AccessGranted
         WHERE IS_ROLEMEMBER(@RoleName) = 1);

GO

EXECUTE AS USER = 'CEO';
SELECT * FROM Sales;
REVERT;
GO

EXECUTE AS USER = 'Fred';
SELECT * FROM Sales;
REVERT;
GO
```

Database User

CEO

Database User

Fred

dbo.Sales - Table

User Name	Country	Sales
Fred	USA	10000
Chris	USA	9500
Tom	France	9600
Fred	Spain	9200
Chris	Germany	9000

User Name	Country	Sales
Fred	USA	10000
Fred	Spain	9200

RLS Lookup Table Example

```
CREATE TABLE RLS.UsersSuppliers (

UsersSuppliersID int NOT NULL CONSTRAINT
PK_RLSUsersSuppliers PRIMARY KEY CLUSTERED
IDENTITY
,UserID nvarchar(255) NOT NULL
,SupplierID int NOT NULL )
--Grant the test user access
--to a single supplier ID

INSERT INTO RLS.UsersSuppliers (UserID,
SupplierID)

VALUES ('RLSLookupUser’,4)
```

UserSuppliersID	UserID	SuppliersID
1	RLSLookupUser	4

```
CREATE FUNCTION RLS.AccessPredicate_SupplierID_PurchasingSuppliers
(@SupplierID int)
RETURNS TABLE
WITH SCHEMABINDING
AS RETURN

SELECT 1 AccessResult
FROM RLS.UsersSuppliers US
INNER JOIN Purchasing.Suppliers PS
  ON US.SupplierID = PS.SupplierID
WHERE US.SupplierID = @SupplierID
AND US.UserID = USER_NAME()
GO

CREATE SECURITY POLICY
RLS.SecurityPolicy_SupplierID_PurchasingSuppliers
ADD FILTER PREDICATE
RLS.AccessPredicate_SupplierID_PurchasingSuppliers
(SupplierID) ON Purchasing.Suppliers,
ADD BLOCK PREDICATE
RLS.AccessPredicate_SupplierID_PurchasingSuppliers(SupplierID) ON
Purchasing.Suppliers WITH (STATE = ON, SCHEMABINDING = ON)
```

RLS Side Attacks

- Malicious Security Policy Manager
- Divide By Zero Attack
- Cross-feature compatibility

Divide By Zero Attack

```
declare @i int = 2995, @CreditLimit int
WHILE @i < 3002
BEGIN
    BEGIN TRY
    SELECT @CreditLImit = 1
    FROM Sales.Customers
    WHERE CustomerID = 801
        AND 1/(@i - CreditLimit) = 0
    END TRY
    BEGIN CATCH
        PRINT 'Bingo Credit Limit is '+ CAST(@i AS VARCHAR(200))
    END CATCH
select @i = @i+1
END
```

100 %

Messages

```
Bingo Credit Limit is 3000
```

Divide By Zero At Scale

Results | Messages

	CustomerID	CreditLimit
1	801	3000
2	802	2940
3	803	2000
4	804	2200
5	805	3300
6	806	3000
7	807	3100
8	808	1800
9	809	1700
10	810	1200
11	811	2100
12	812	2200
13	813	2600
14	814	2310
15	815	2900

```
WHILE @CustomerID < 850
BEGIN
   WHILE @CreditLimit < 10000 AND @CustomerID NOT IN (SELECT
CustomerID from @Customers)
   BEGIN
        BEGIN TRY
            INSERT INTO @Customers (CustomerID, CreditLimit)
            SELECT C.CustomerID, C.CreditLimit
            FROM Sales.Customers C
            WHERE C.CustomerID= @CustomerID
          AND 1/(C.CreditLimit - @CreditLimit) = 0
        END TRY
        BEGIN CATCH
             INSERT INTO @Customers (CustomerID, CreditLimit)
             SELECT @CustomerID, @CreditLimit
             BREAK --Stop processing this row when the correct value
is found

        END CATCH
        SELECT @CreditLimit += 1
  END
  SELECT @CustomerID += 1
  SELECT @CreditLimit = 0
END

SELECT *
FROM @Customers
ORDER BY CustomerID
```

ProcureSQL

RLS Identify Attacks

Excessive Errors – 8134 (Divide By Zero)

SQL Server Side Trace

Extended Events

Server or Database Audits (Why we started with Audits ☺)

SQL Advanced Threat Protection

Performance changes

Excessive CPU Usage

Excessive requests per second

RLS Best Practices

It's highly recommended to create a separate schema for the RLS objects: predicate functions, and security policies.

The security policy manager doesn't require SELECT permission on the tables they protect.

Keep predicate functions short and sweet. Avoid using excessive table joins in predicate functions to maximize performance.

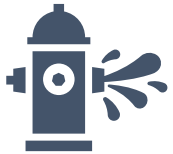
Follow regular performance tuning best practices for predicates.

Features That Don’t Play Well With RLS

- DBCC SHOW_STATISTICS
- Filestream (Not Supported)
- Memory-Optimized Tables
- Indexed Views
- Change Data Capture
- Full-Text Search
- Columnstore Indexes
- Partitioned Views
- Temporal Tables

More Details : Microsoft Learn RLS Cross-Feature Compatibility

RLS Anti-Patterns



Using Features that can introduce data leakage

Highly Transactional Systems

Databases without Direct user access

Less Experienced Teams

Staging or Loading Tables

ProcureSQL

Always Encrypted

Why Should We Use Always Encrypted?

Feature	Encrypt Data At	Protects Against	Key Management	App Changed Needed
Transparent Data Encryption (TDE)	Rest (files)	Physical theft	Internal or EKM	No
Backup Encryption	Backup files	Backup theft	Internal or EKM	No
Column Level Encryption	Column Level	Unauthorized access to column data	Internal or EKM	Yes
TLS	Transit	Network eavesdropping	N/A	No
Always Encrypted	Column Level	DBAs, admins, memory attacks	External (client side)	Yes

Encryption Keys

```
CREATE COLUMN MASTER KEY [CMK1]
WITH
(
KEY_STORE_PROVIDER_NAME =
N'MSSQL_CERTIFICATE_STORE',
KEY_PATH =
N'LocalMachine/My/2379554....',
ENCLAVE_COMPUTATIONS (SIGNATURE =
0x5B1A... )
```

```
COLUMN ENCRYPTION KEY [CEK1]
WITH VALUES
(
 COLUMN_MASTER_KEY = [AE_CMK1],
ALGORITHM = 'RSA_OAEP',
ENCRYPTED_VALUE = 0x01700000016...
)
```

Two-level key hierarchy

- Column encryption keys (CEK) – encrypts data
- Column master keys (CMKs) – encrypts CEKs

The database stores metadata about keys

- Enclave-enabled – CMKs have ENCLAVE_COMPUTATIONS set
- Enclave-enabled CEKs are encrypted with enclave-enabled CMKS

Key Storage Decisions

Windows – Certificate Store

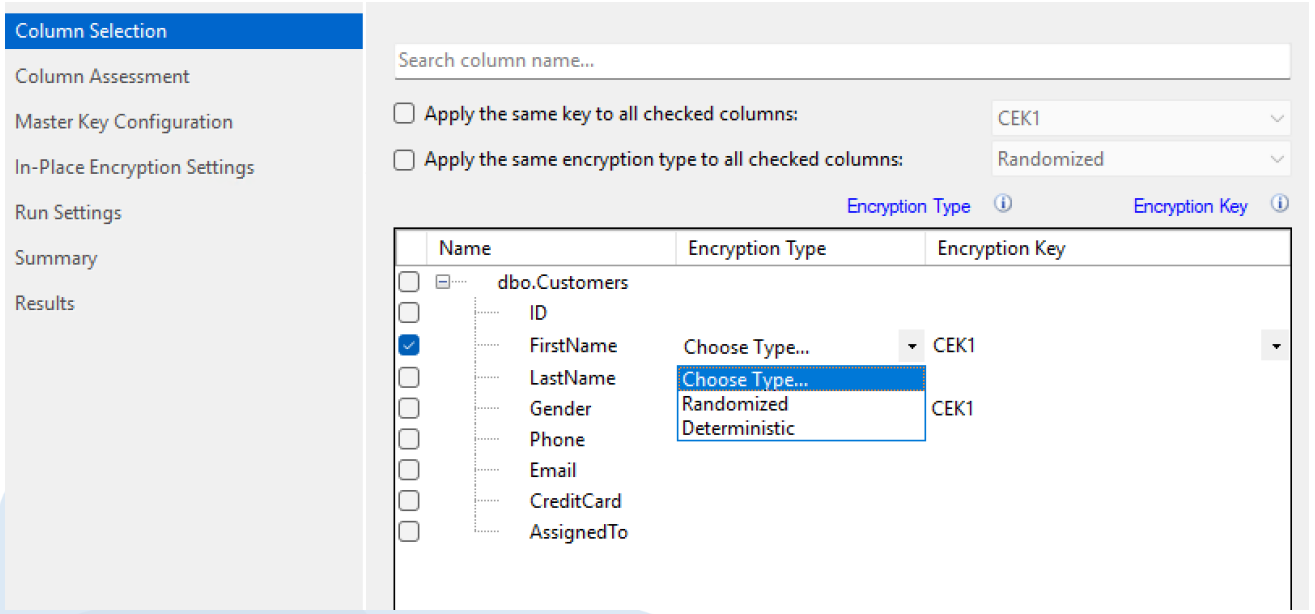
- Control and Compliance
- No Cloud Dependency
- Existing Infrastructure Utilization
- Simple Implementation for Smaller Deployments
- No Additional Costs

Azure Key vault

- Requires Azure Key Vault Access
- Separation of Duties
- End to End Protection
- Centralized Management
- Scalable and Flexibility
- Enhanced Security (RBAC)
- Simple Key Rotation

Always Encrypted Types in the Beginning

- **Deterministic**
  - Less Secure and Predictable
  - Great WHERE clause equality JOINS
  - Indexes
- **Randomize**
  - More Secure
  - **Non-Serchable (SQL 2016 days)**



<u>You must decide during the setup of encrypting a column.</u>

Deterministic vs Randomized

Plaintext

Results | Messages

	FirstName	Gender
1	Gail	F
2	Terri	F
3	Diane	F
4	Ken	M
5	Roberto	M
6	Rob	M
7	Jossef	M
8	Dylan	M

Deterministic (Not Random)

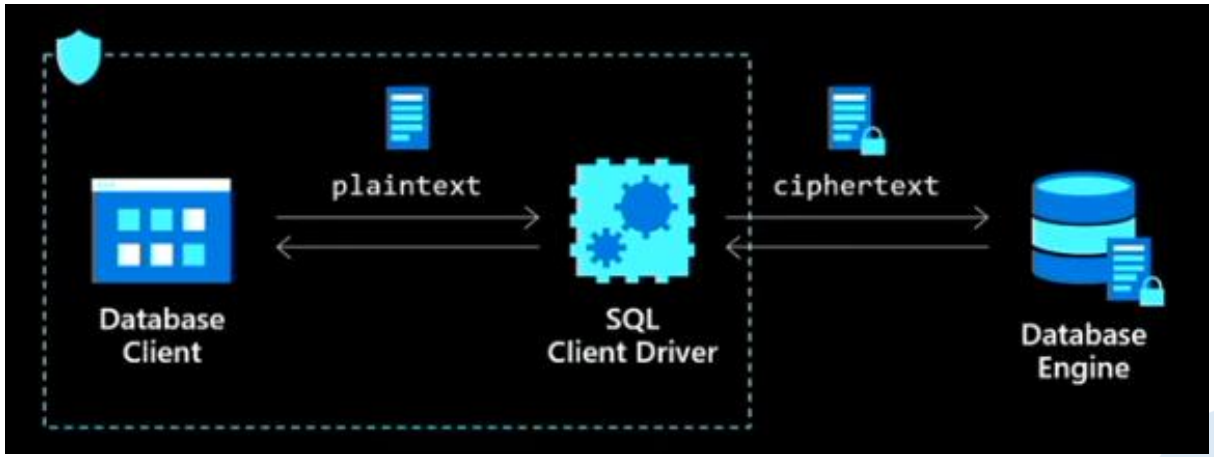
	FirstName	Gender
1	Gail	0x01D735B667AFDA97527CBF1B78F7E491932C5D71C99535AA99...
2	Terri	0x01D735B667AFDA97527CBF1B78F7E491932C5D71C99535AA99...
3	Diane	0x01D735B667AFDA97527CBF1B78F7E491932C5D71C99535AA99...
4	Ken	0x0185F5C2FBCA71111F480EC98AD316036D0AB93F40CEFD249F...
5	Roberto	0x0185F5C2FBCA71111F480EC98AD316036D0AB93F40CEFD249F...
6	Rob	0x0185F5C2FBCA71111F480EC98AD316036D0AB93F40CEFD249F...
7	Jossef	0x0185F5C2FBCA71111F480EC98AD316036D0AB93F40CEFD249F...
8	Dylan	0x0185F5C2FBCA71111F480EC98AD316036D0AB93F40CEFD249F...

Randomized

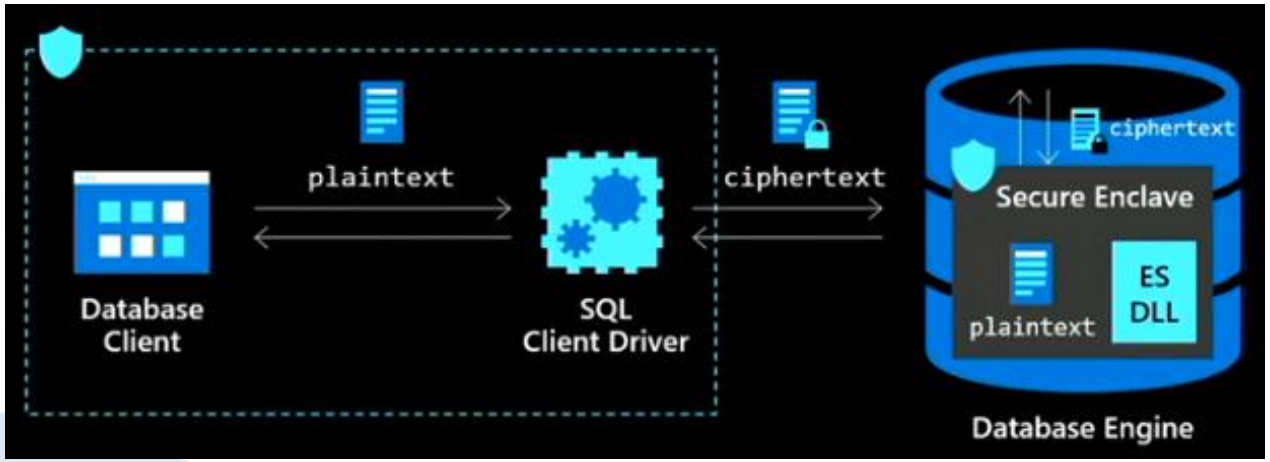
	FirstName	Gender
1	Gail	0x01E2D1CDB2DB0C5D25AB7349A4A586B55907C310803A7F0...
2	Terri	0x01A77DEBC67840757FF4F75D8053D44ECCD5AA7E235AFA8...
3	Diane	0x01BEA66C9E94D5952CBF17124CD5C02D12473A8BCA61E81...
4	Ken	0x01FC1B79A0C83334384EBBCD820E2B4FDFB9B13AD888B83...
5	Roberto	0x010C9B87ED821E0292B6CBF3AAF0DE7578792A79415319E...
6	Rob	0x0190F8ADFD36E2B97D8608F8B3F2DA102776EB85DD3CD4...
7	Jossef	0x01BA22B1D9154B4D23A85F1494949512678E3A62237E9A8C...
8	Dylan	0x014A5FAB36036F65BAF7EE3AF3F68678DF1858A7D00BADE...

Always Encrypted with Secure Enclaves

AE without Secure Enclaves



AE with Secure Enclaves



[SQL Bits – Demystifying Always Encrypted with security enclaves]

Computation over encrypted Columns

Randomized

No scalar operations

Deterministic

Equality queries

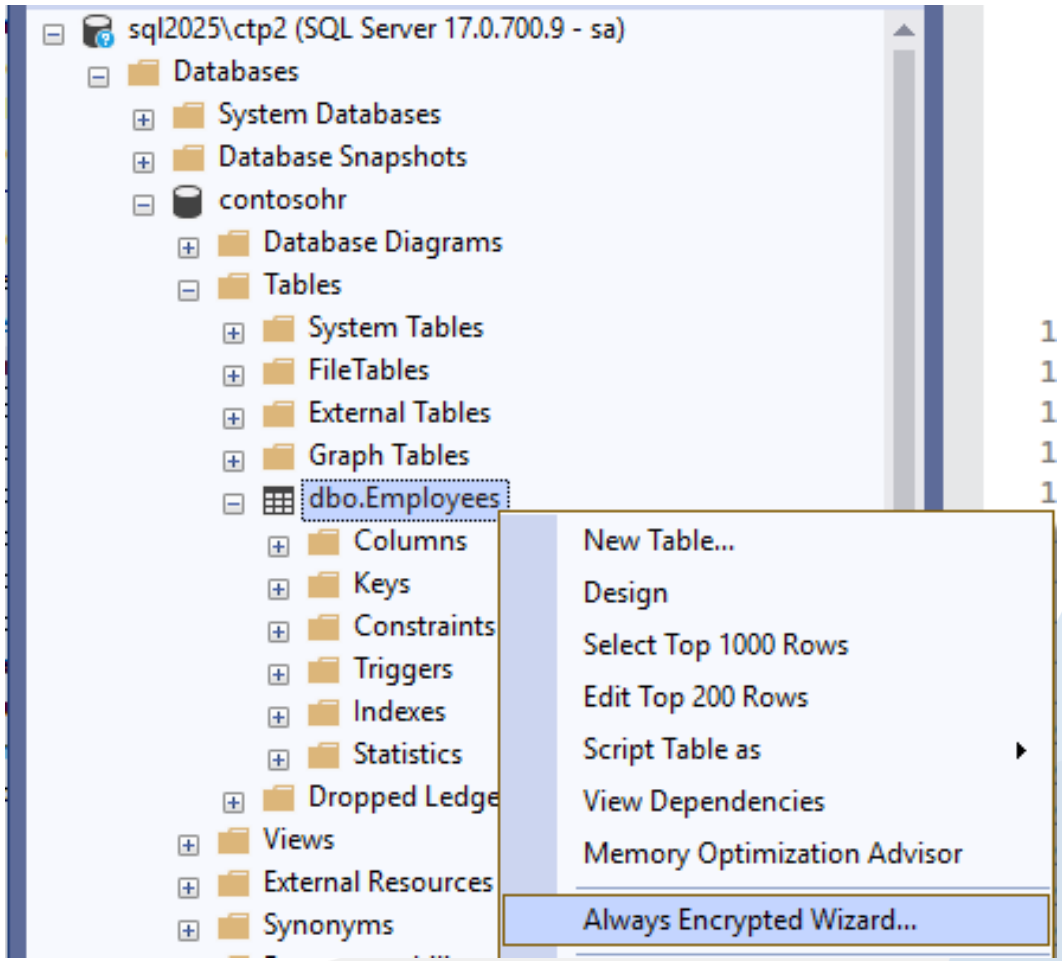
Randomized & enclaved-enabled keys

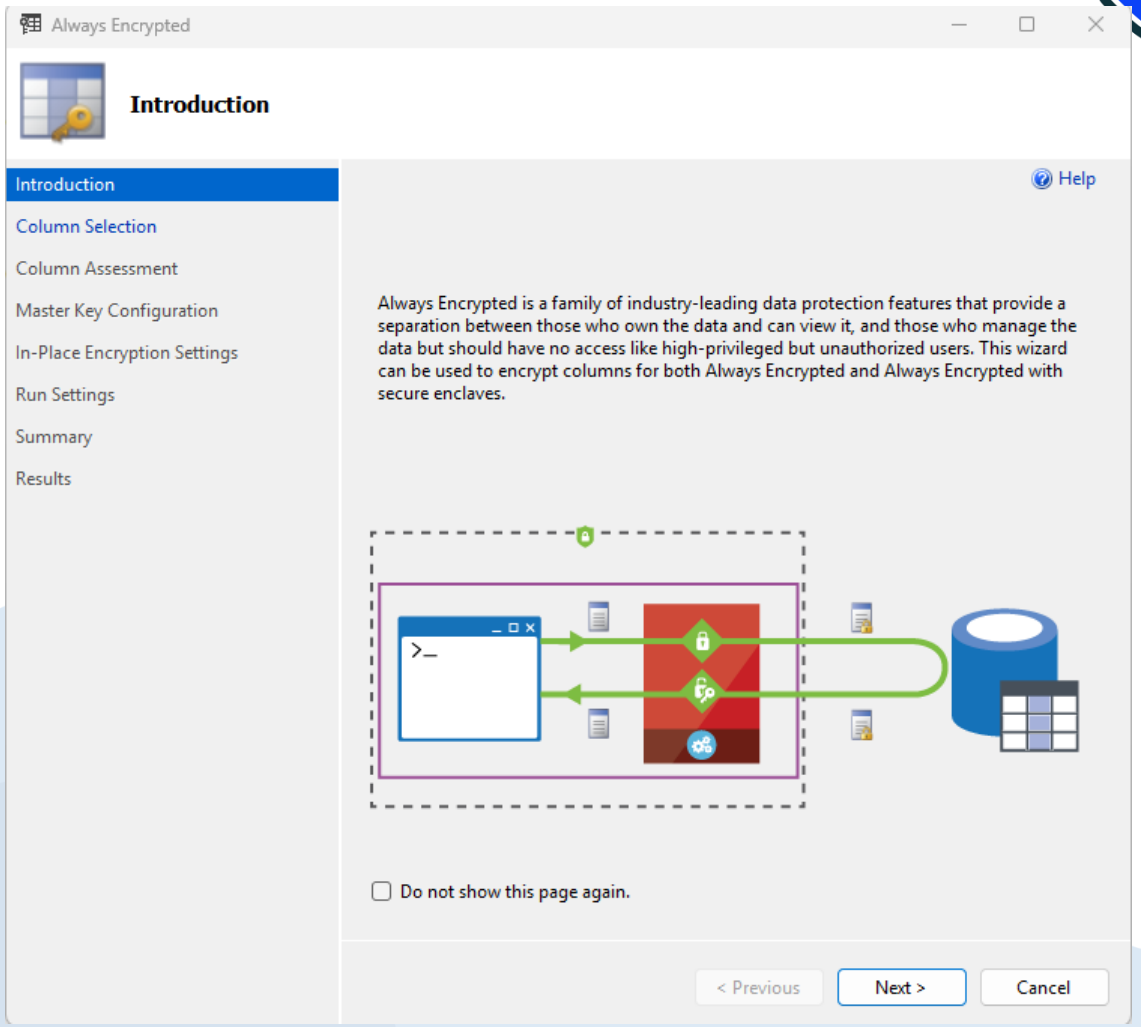
Range/LIKE queries, sorting

Always Encrypted - Required Code Changes

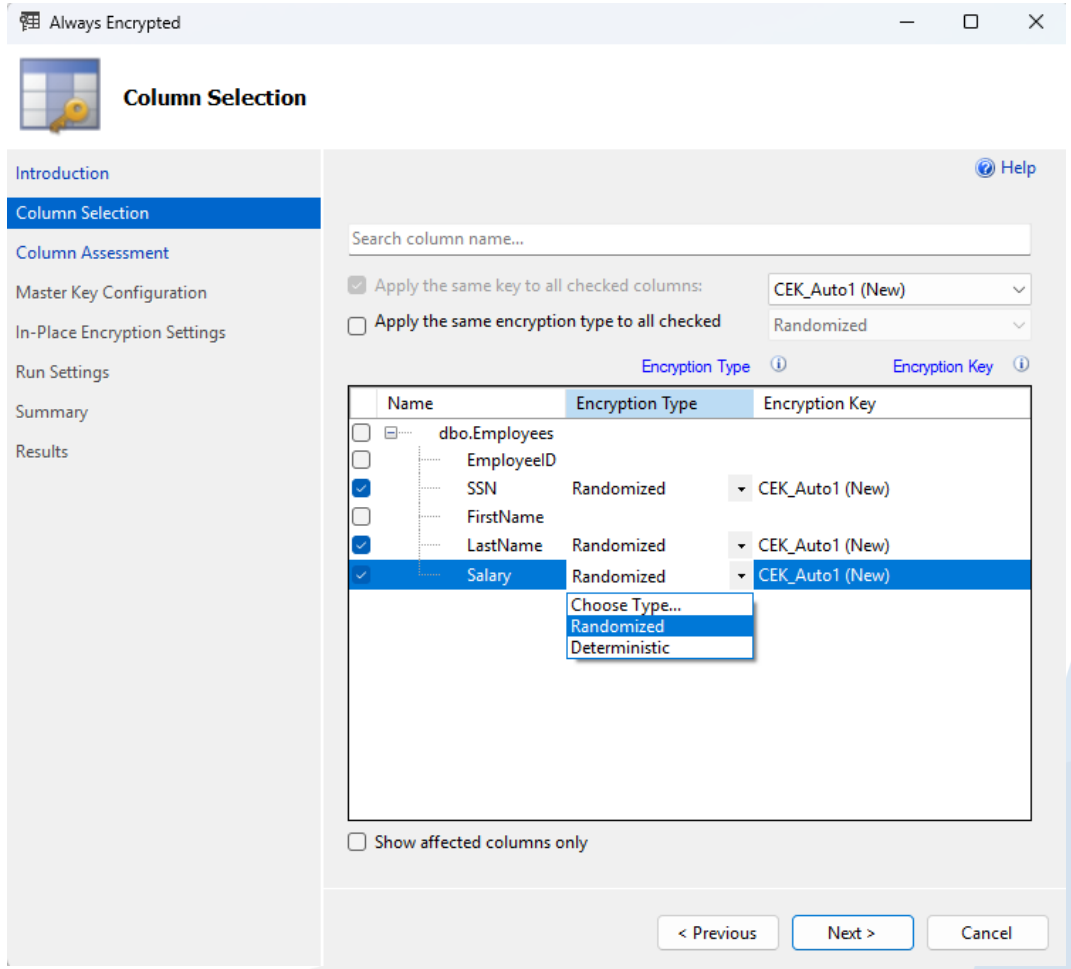
- Connection String (Encrypted Setting=Enabled)]
- Parametrization of Queries – No literals in filters
- Explicit Data Type (Parameter Type must match encrypted column type)

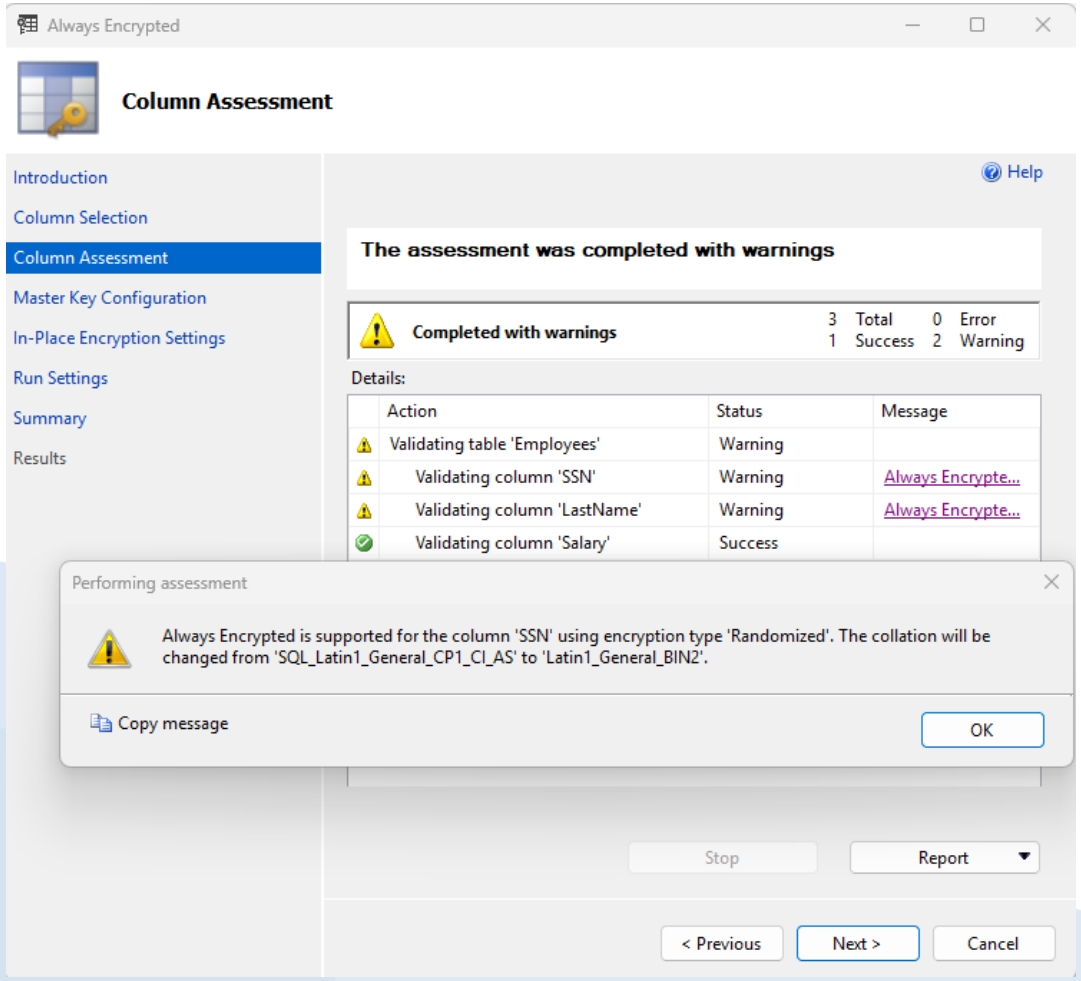
Enabling Always Encrypted (1 of 5)



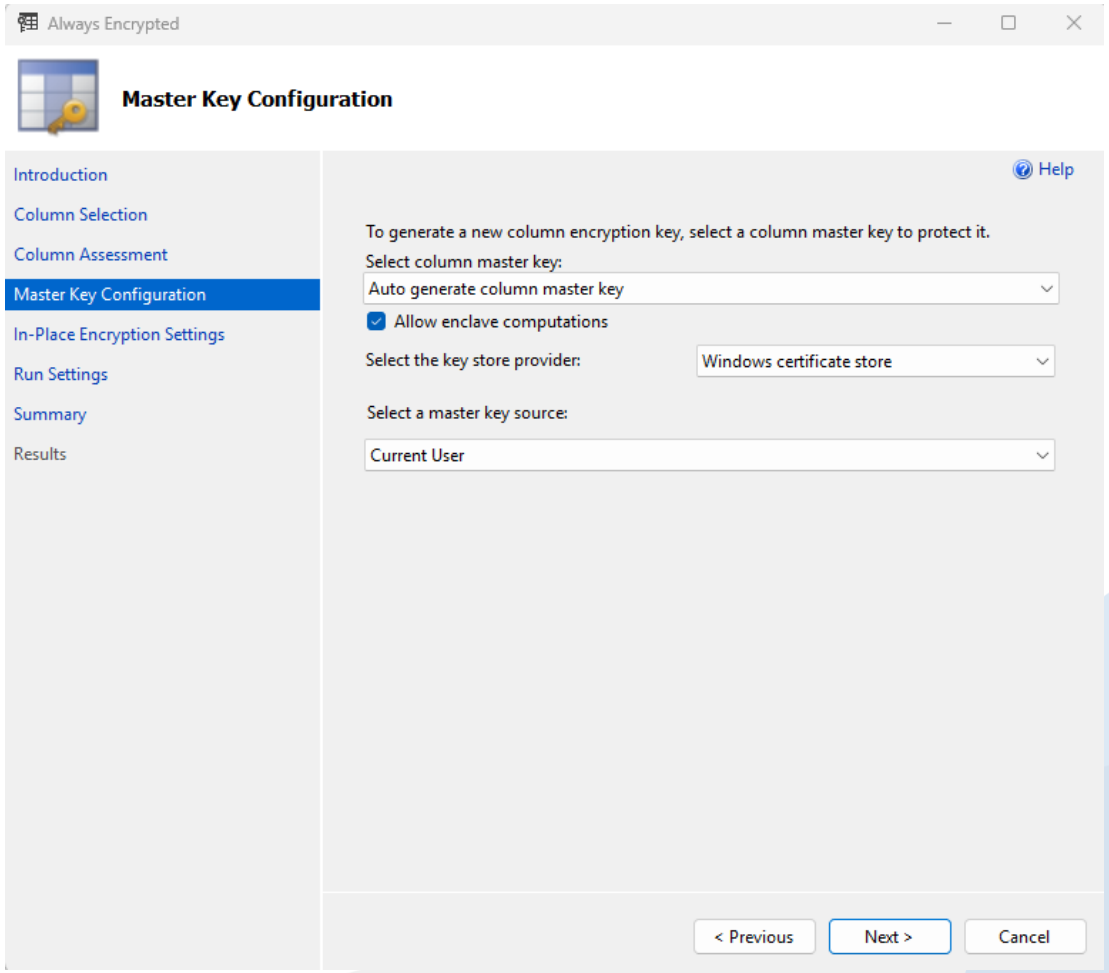


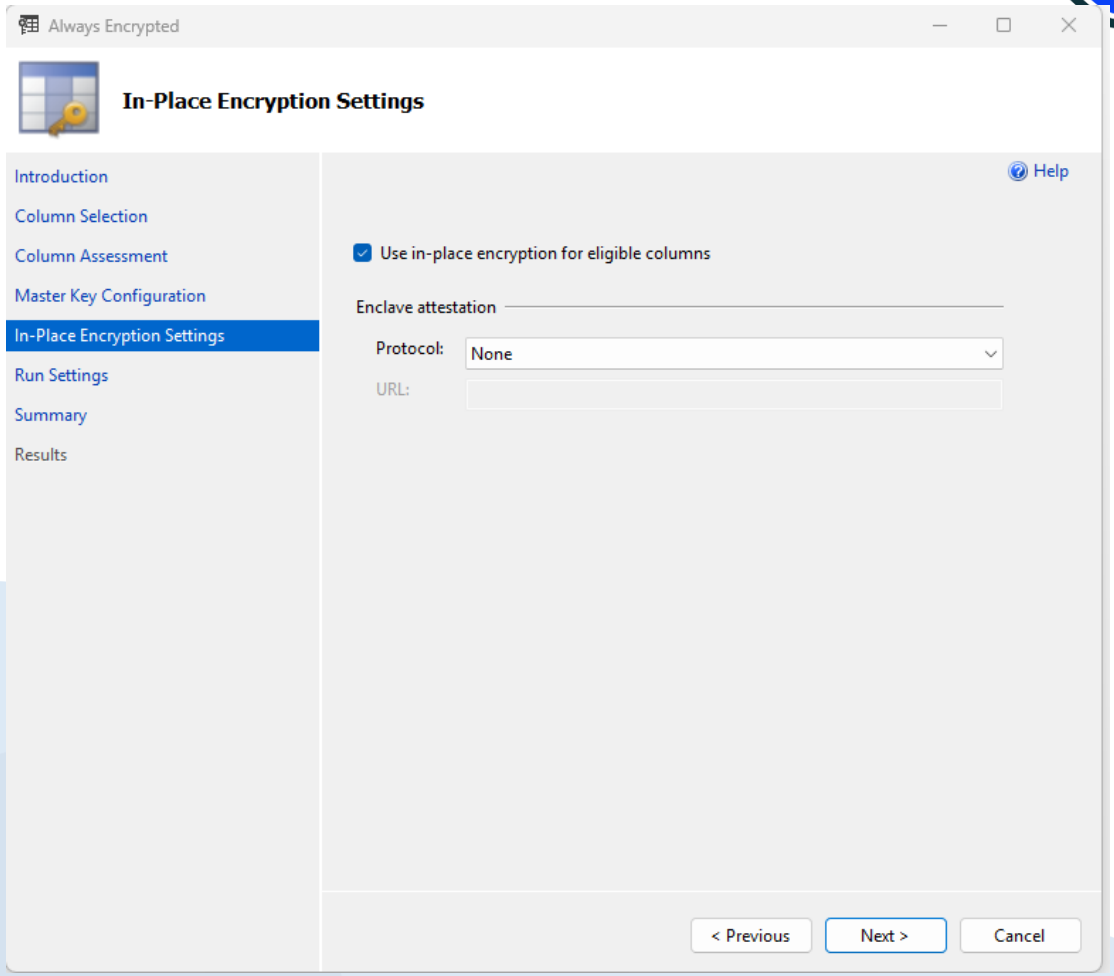
Enabling Always Encrypted (2 of 5)



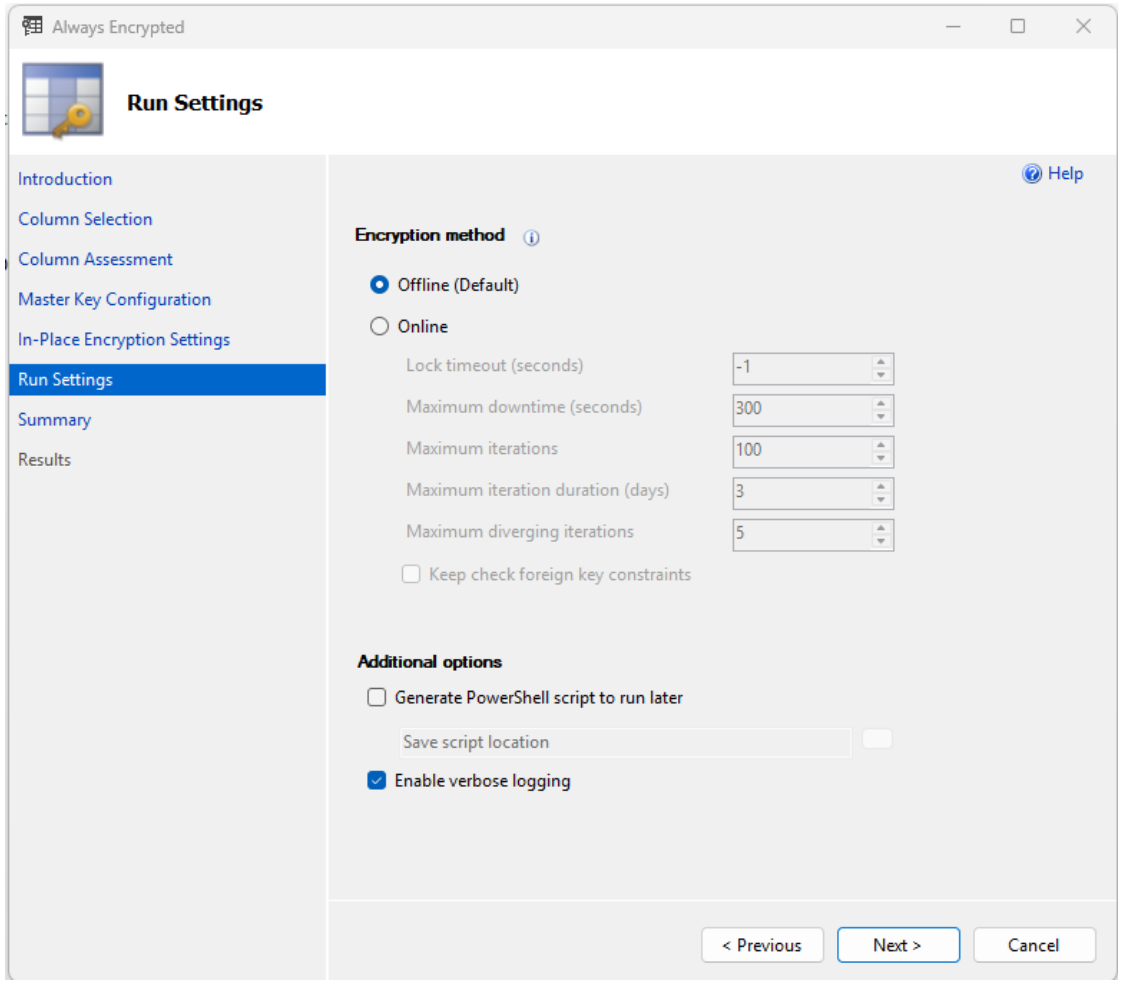


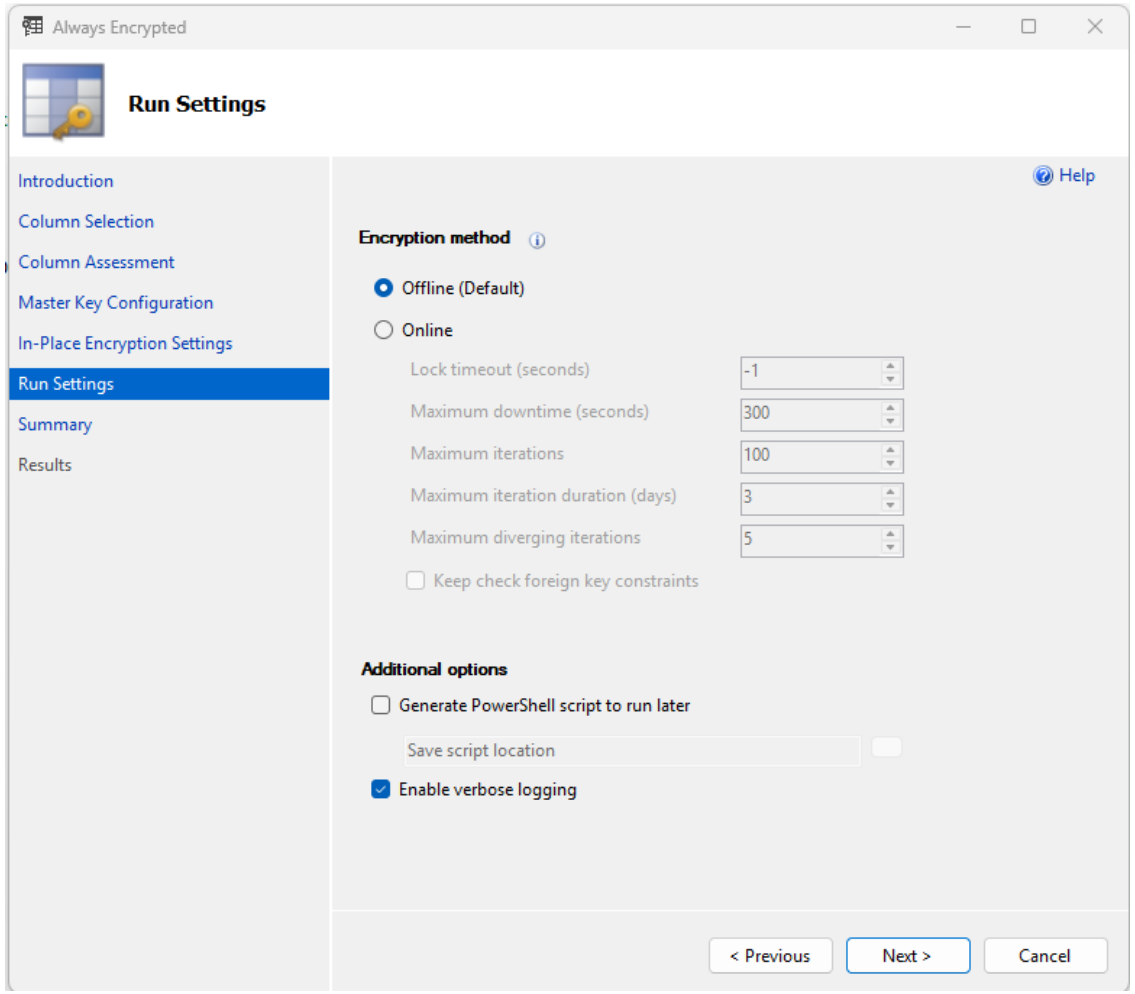
Enabling Always Encrypted (3 of 5)





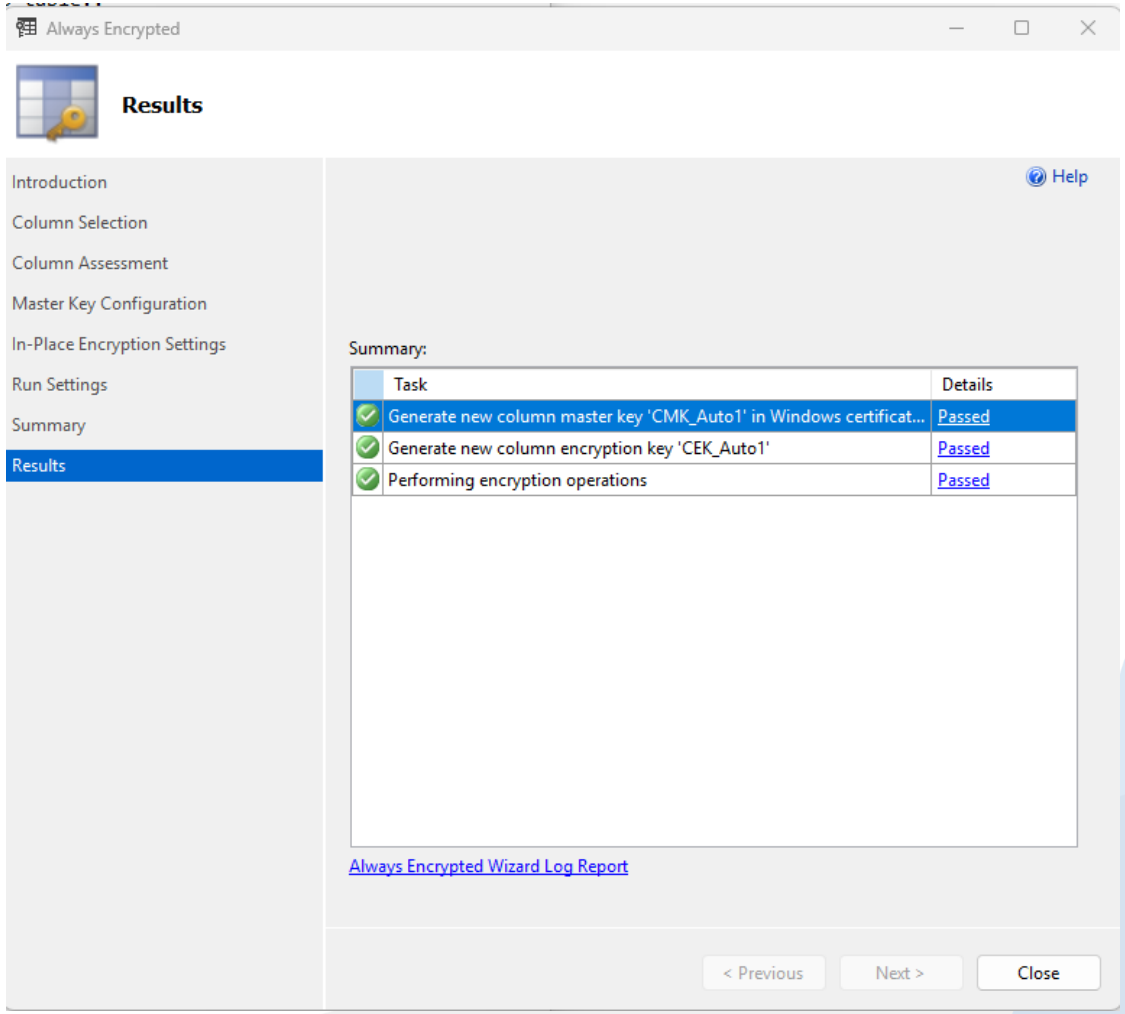
Enabling Always Encrypted (4 of 5)

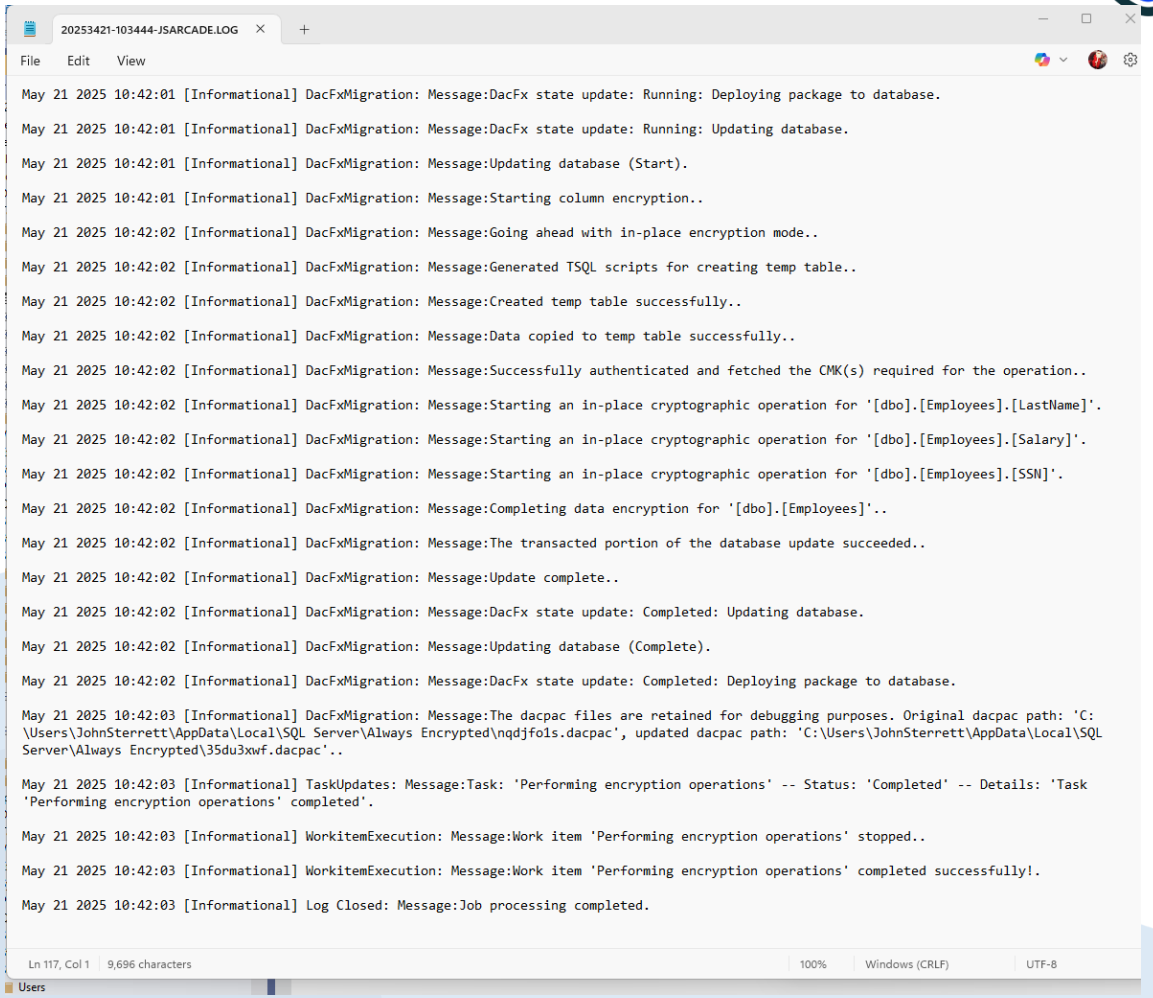




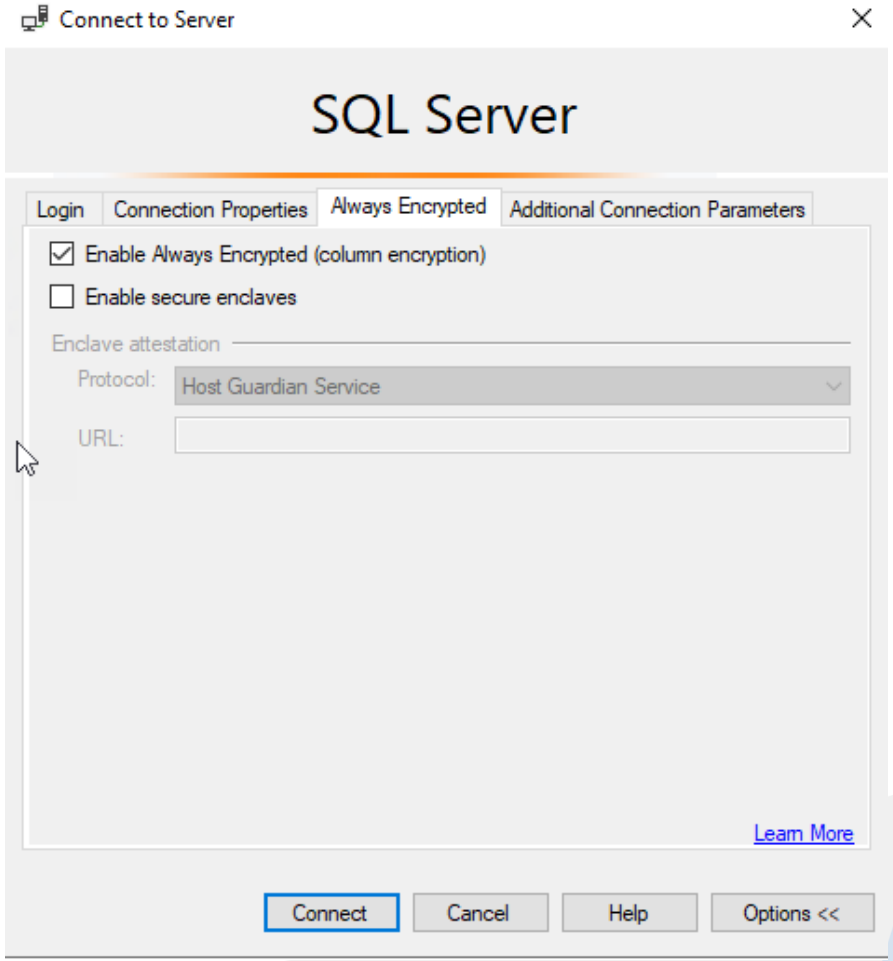
ProcureSQL

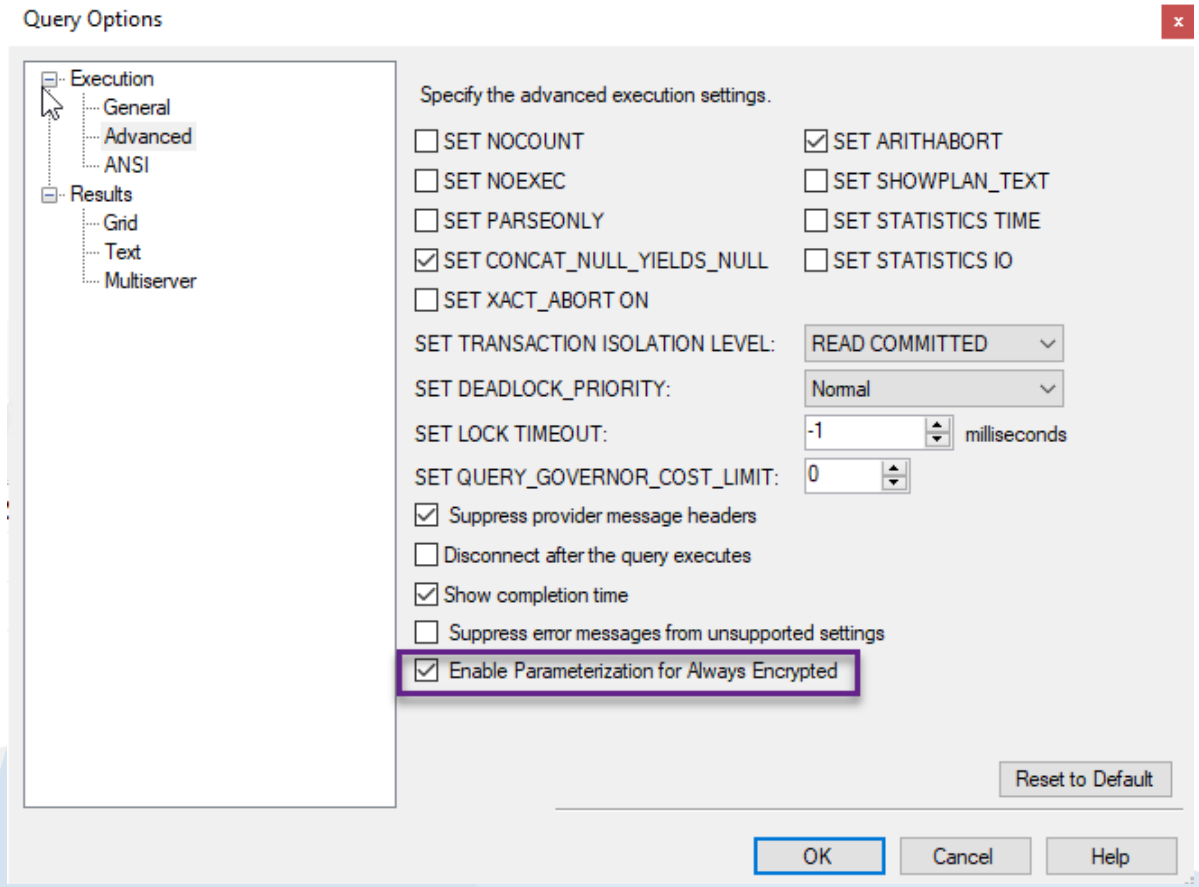
Enabling Always Encrypted (5 of 5)





Connecting with SSMS 21





Encrypted Columns – No keys

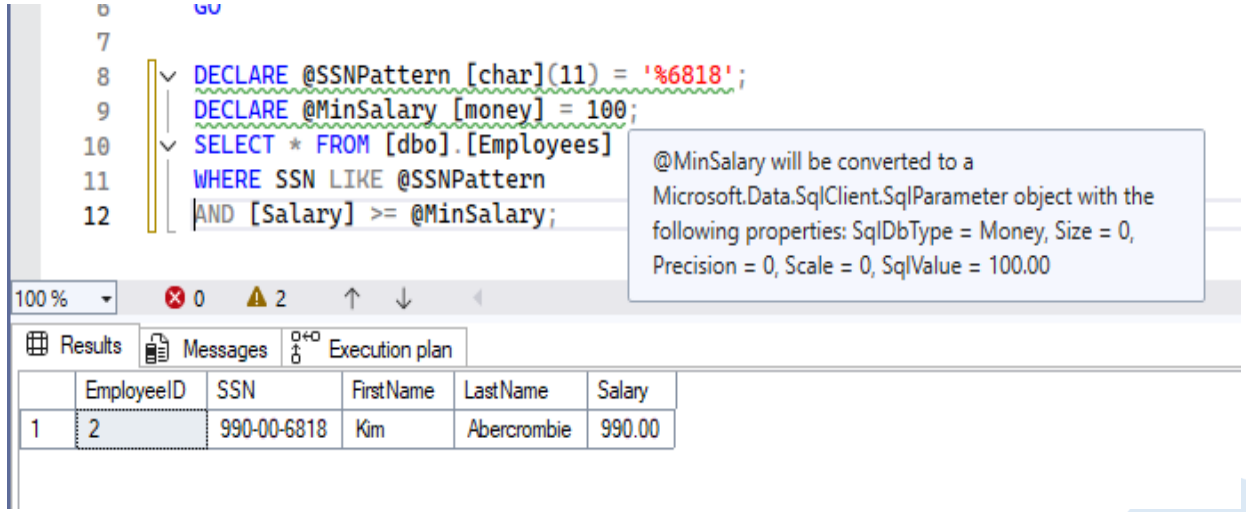
```
SELECT [EmployeeID]
      ,[SSN]
      ,[FirstName]
      ,[LastName]
      ,[Salary]
  FROM [WideWorldImporters].[HR].[Employees]
```

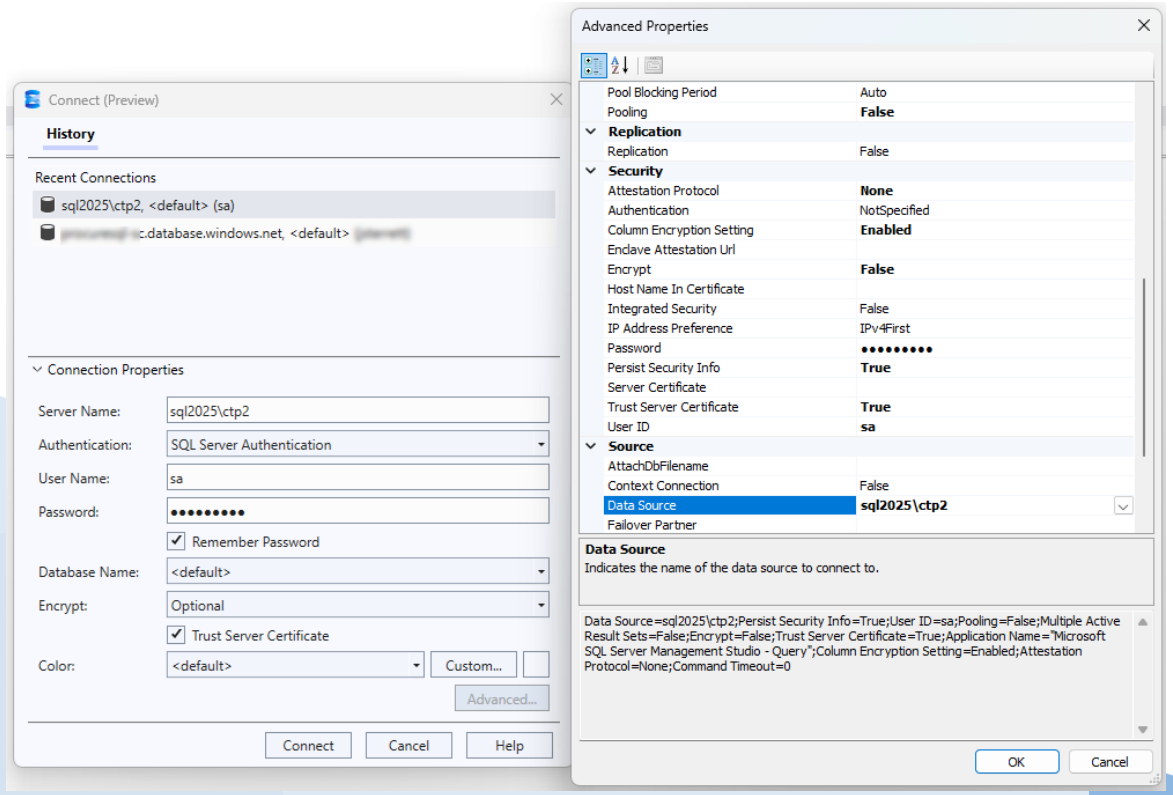
100 %

Results | Messages

	EmployeeID	SSN	FirstName	LastName	Salary
1	1	0x019990EFFD7BB0BFF94F4B5...	Catherine	Abel	0x01FD581DD2F6D7578...
2	2	0x014D1454895116660A67D9...	Kim	Abercrombie	0x016719580C6B27A9C4...

Decrypted Data – With Keys





Questions and Contact Information

Feedback for slides

Microsoft CERTIFIED TRAINER MCT

john@procuresql.com

procuresql.com

https://www.linkedin.com/in/johnsterrett