Session 19

SQL Server 2016

Security Upgrades and Working with JSON

Objectives

- Explain how to use SQL Server features to protect data at rest and in motion
- Describe security enhancements in SQL Server 2016
- Explain how to work with JSON data

Security Enhancements of SQL Server 2016 1-2

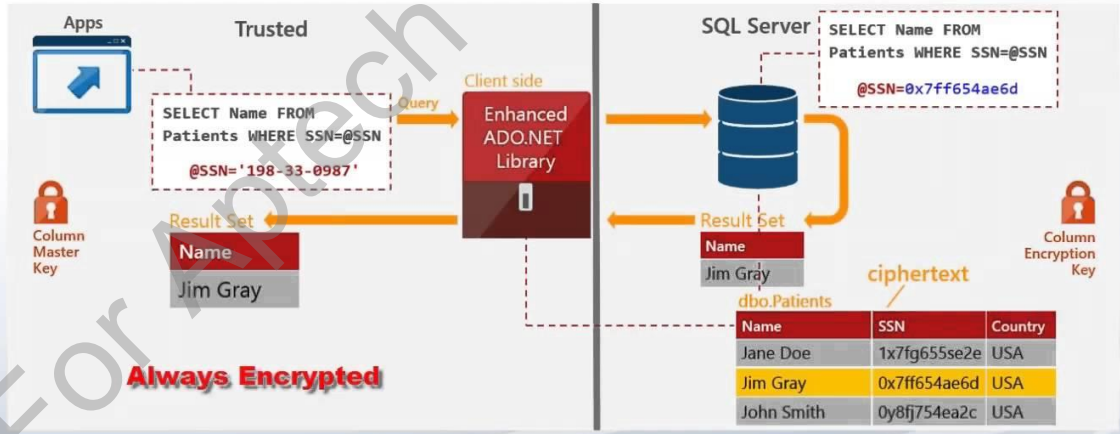
- Always Encrypted
- Dynamic Data Mask
- Transparent Data Encryption
- Row–Level Security

Security Enhancements of SQL Server 2016 2-2

Following figure represents the Security Enhancements of SQL Server 2016:

Improvements on Security

- ✓ **Always Encrypted: C**olumns are encrypted using keys on the application driver side
- ✓ **Dynamic Data Masking:** Establishes policies to mask sensitive column values to specific users or roles
- ✓ **Row Level security:** Establishes policies that filter our specific rows based on the user role



Always Encrypted 1-2

Always Encrypted is a feature that encrypts sensitive data inside client applications.

The two types of encryption supported by Always Encrypted are:

- Randomized Encryption
  - Encryption is more secure
  - Data encrypted in a randomized manner
- Deterministic Encryption
  - Same encrypted value generated for any specific plain text value

Always Encrypted 2-2

Two types of keys used by Always Encrypted are:

Column Master Keys	Column Encryption Keys
• Master keys protect column encryption keys	• Sensitive data stored in database is encrypted with these types of keys

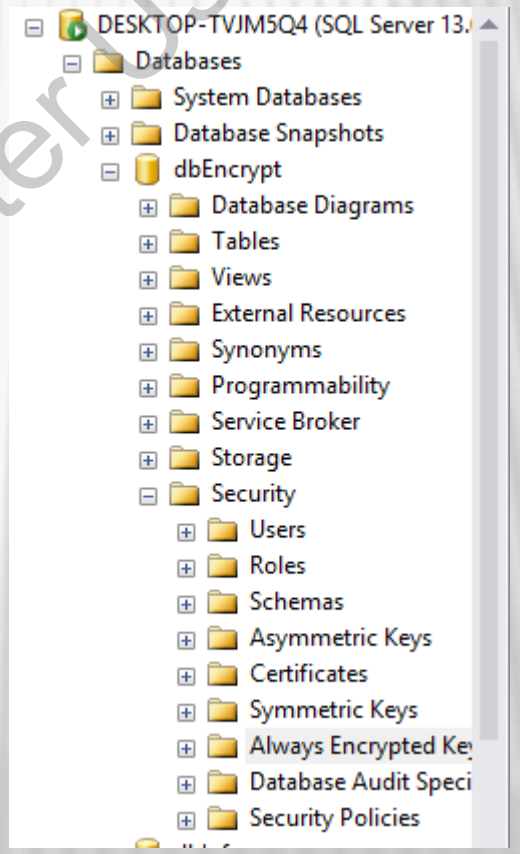
Getting Started with Always Encrypted 1-8

SQL Server 2016

Step-by-step procedure to encrypt a database with Always Encrypted:

1. Expand database in SSMA, expand security, and then create Column Master key

Always Encrypted Keys item can be seen as shown:



Getting Started with Always Encrypted 2-8

2. Expand 'Always Encrypted Keys' option

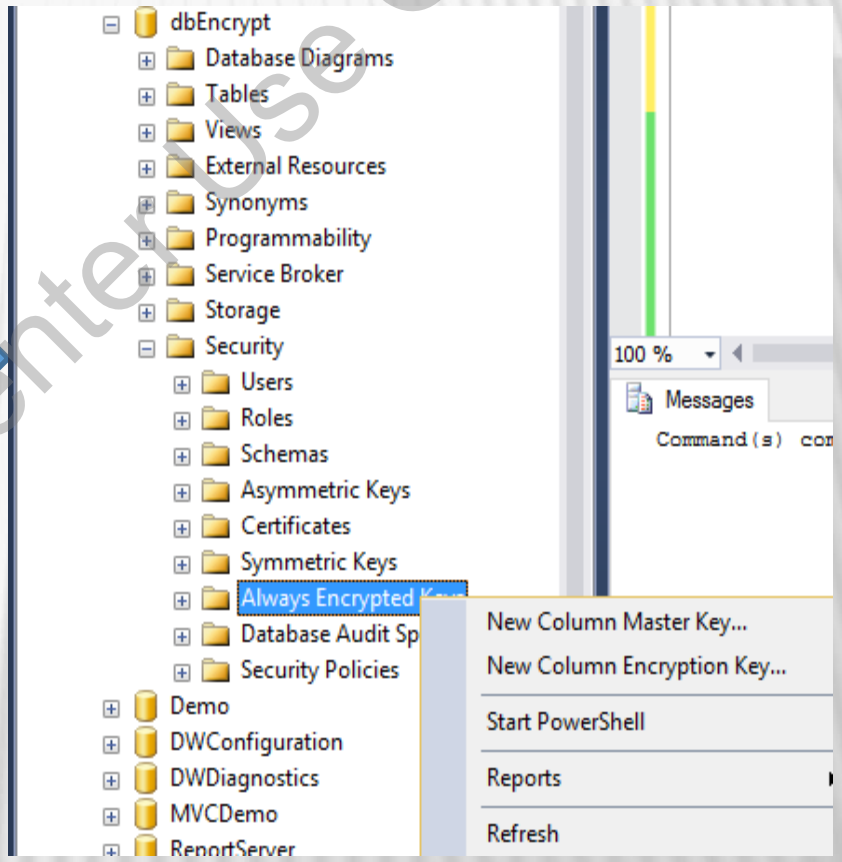
This gives more options as shown:

Getting Started with Always Encrypted 3-8

SQL Server 2016

3. To create Column Master Key, click the 'Column Master Keys' item

This displays options as shown:

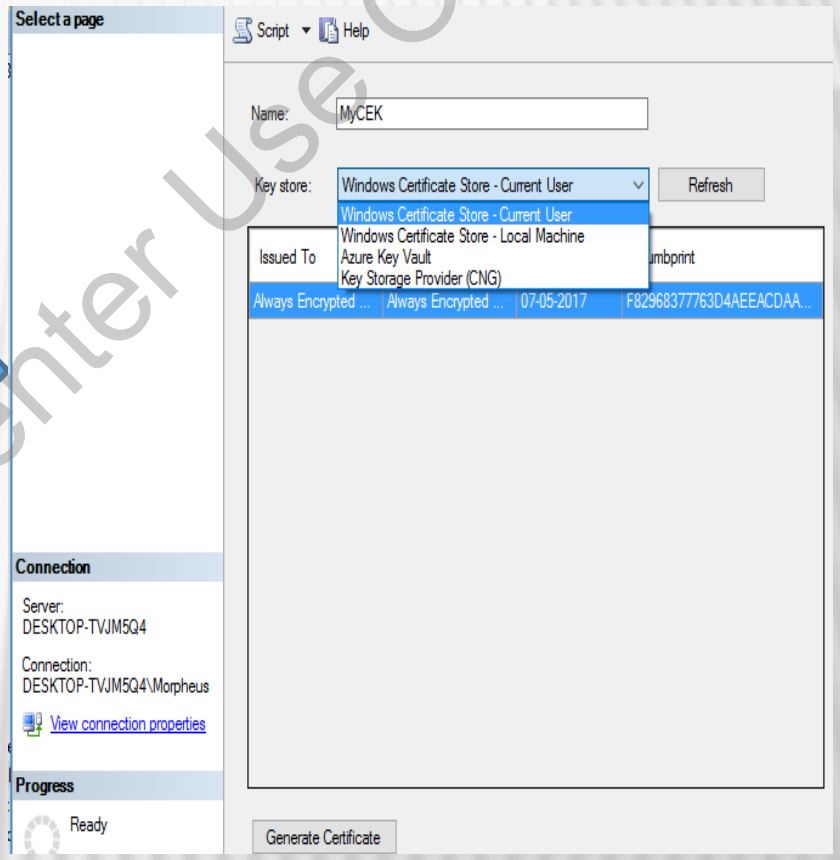


Getting Started with Always Encrypted 4-8

SQL Server 2016

4. Click 'New Column Master Key' item in the drop-down list.

Screen is displayed as shown:

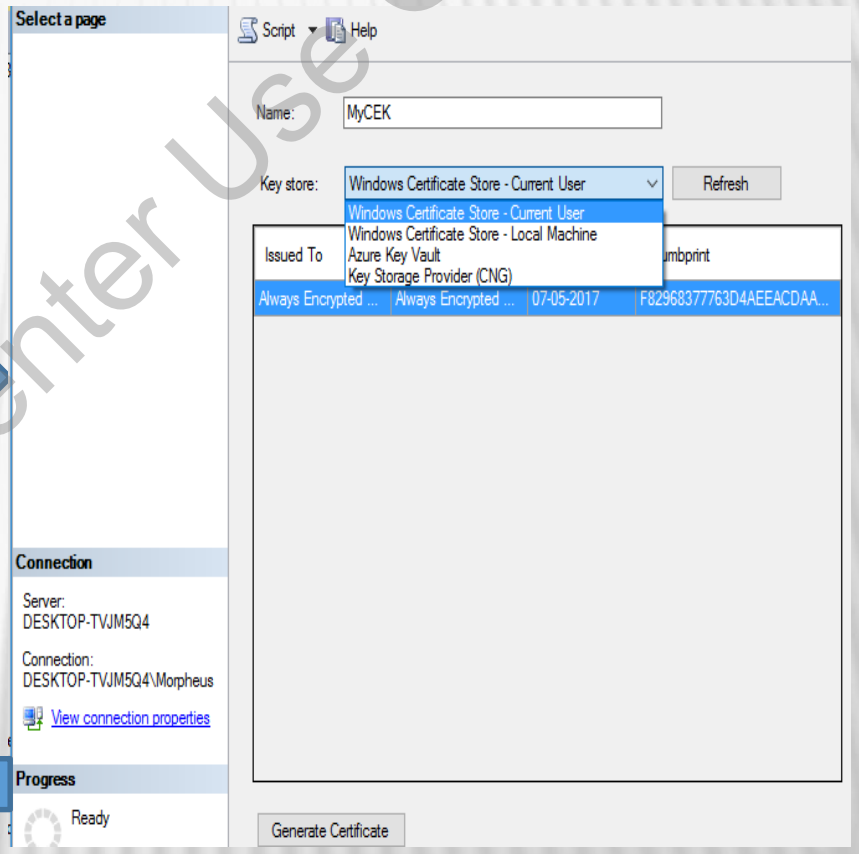


Getting Started with Always Encrypted 5-8

SQL Server 2016

5. Enter a name for Column Master Key in the name section. Expand 'Key store' drop-down box.

Different key store options are displayed as shown:



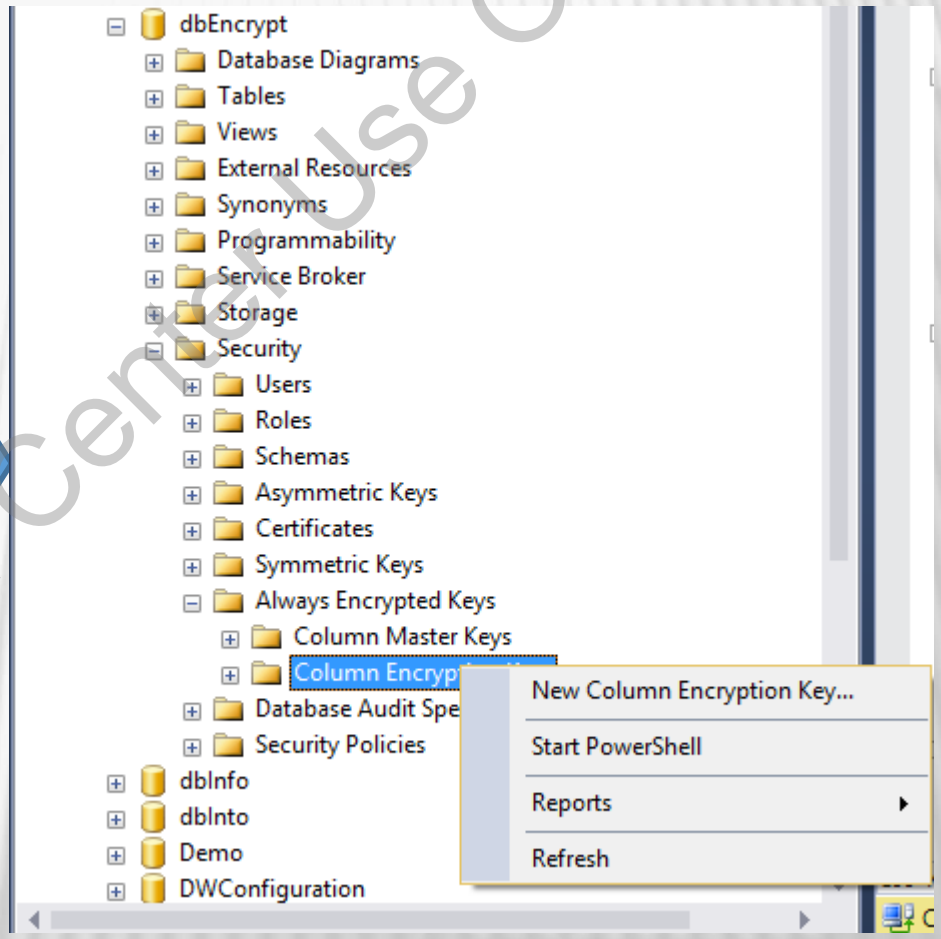
The encrypted master key 'MyCEK' has been created.

Getting Started with Always Encrypted 6-8

SQL Server 2016

6. To create a column encryption key, right-click the 'Column Encryption Key' item in the Object Explorer and then select 'New Column Encryption Key'

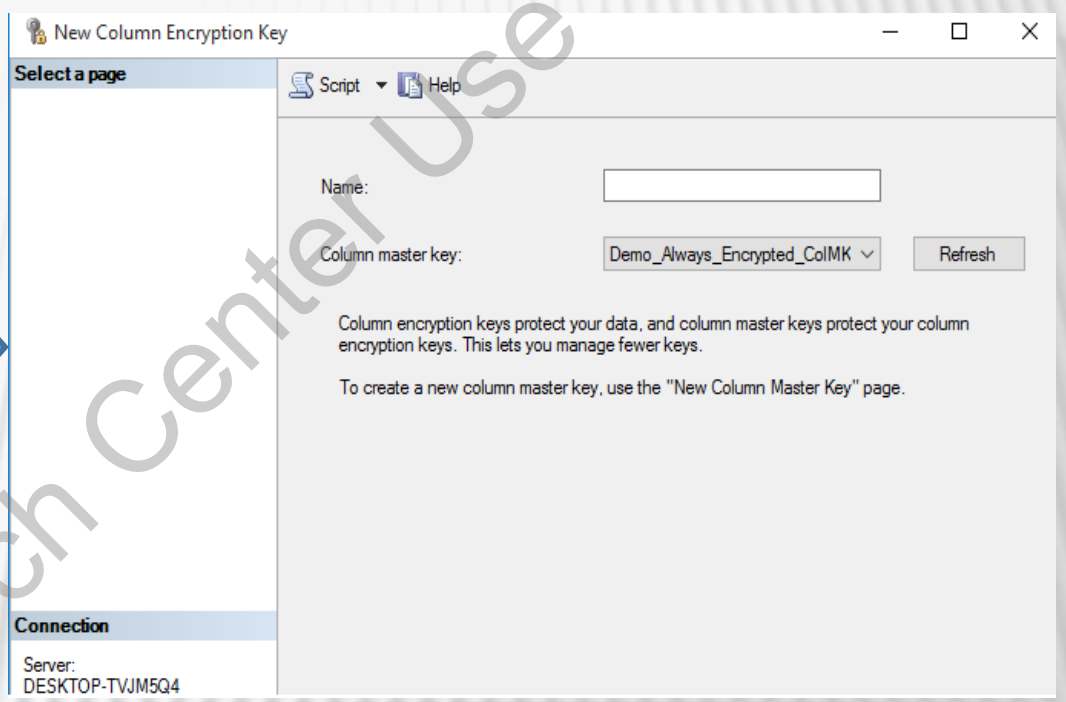
The screen is displayed as shown:



Getting Started with Always Encrypted 7-8

SQL Server 2016

7. Then the 'New Column Encryption Key' is displayed



Getting Started with Always Encrypted 8-8

8. Enter the name of new 'Column Encryption Key' , which is 'Demo_Always_Encrypted_CEK'. Select 'Column Master Key' from drop-down menu, identify the name and master key, then press OK to create column encryption key

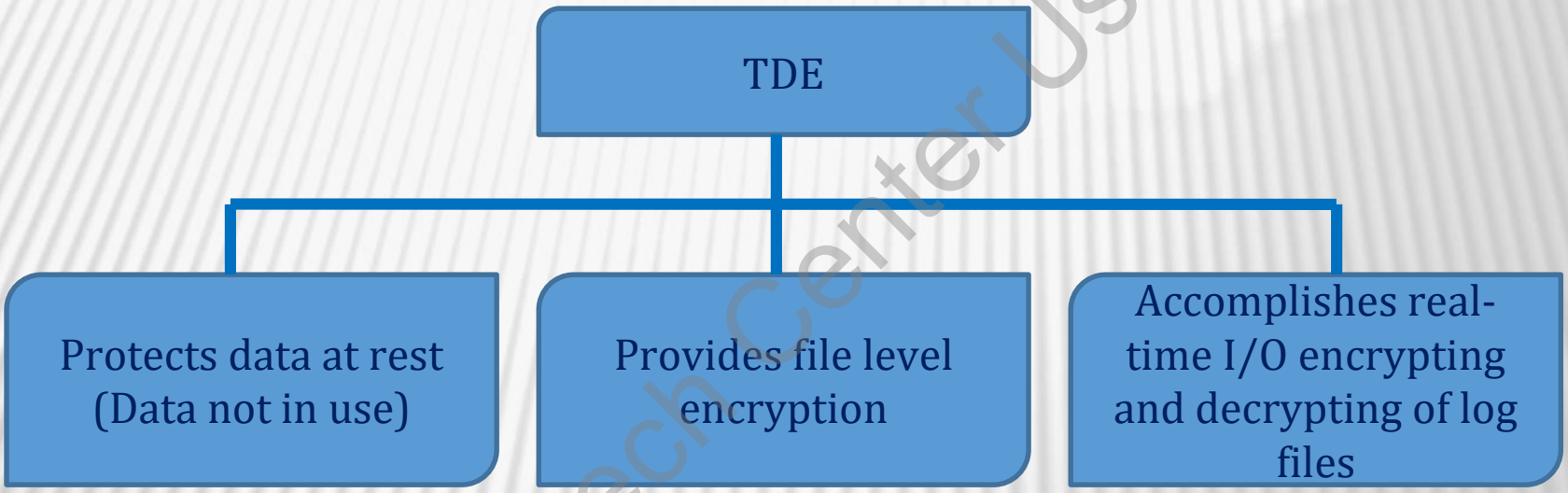
9. Create a new database. Then, create one or more tables with columns to be encrypted

```
CREATE DATABASE dbEncrypt
USE dbEncrypt

CREATE TABLE Employees
(
EmpName nvarchar(60),

COLLATE Latin1_General_BIN2 ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY
Demo_Always_Encrypted_ColMKey,
ENCRYPTION_TYPE = RANDOMIZED,
ALGORITHM =
'AEAD_AES_256_CBC_HMAC_SHA_256'),
                UID varchar(11)
COLLATE Latin1_General_BIN2 ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY =
Demo_Always_Encrypted_ColMKey,
                ENCRYPTION_TYPE =
DETERMINISTIC,
                ALGORITHM =
'AEAD_AES_256_CBC_HMAC_SHA_256'),
                Age int NULL
);
GO
```

Transparent Data Encryption

SQL Server 2016

A technique used to encrypt the SQL Server and Azure SQL database data files.



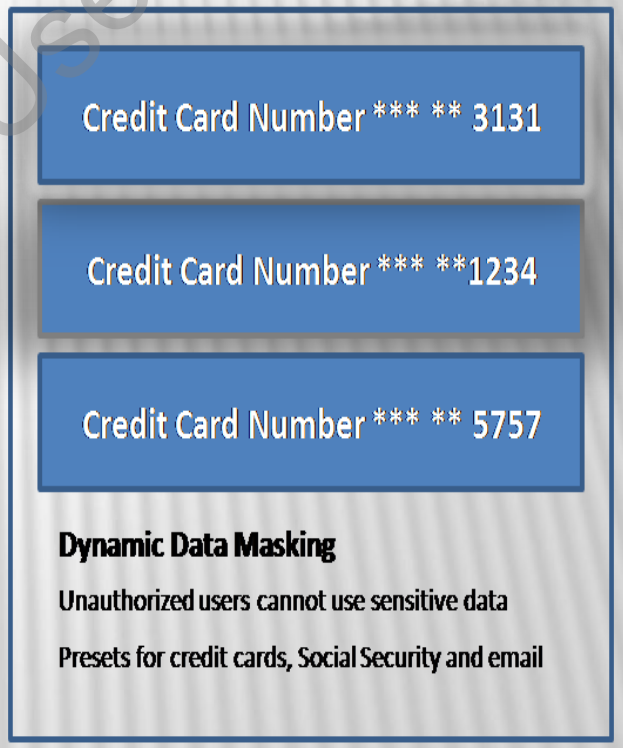
Oracle and Microsoft have adapted TDE technology to encrypt database files.

Dynamic Data Masking 1-6

SQL Server 2016

Dynamic Data Masking is a feature for protecting sensitive data.

- Prevents data from unauthorized access.
- Hides the sensitive query result set and the keeps the database unchanged.
- Sensitive data can be masked without changing existing queries.
- Use DDM in co-ordination with other security features such as auditing, encryption, and row-level security.



Dynamic Data Masking 2-6

Limitations and restrictions of DDM

The column types that cannot have a masking rule defined on them are:

- Encrypted columns
- Filestream
- COLUMN_SET
- Sparse column that is part of a COLUMN_SET
- Computed column
- FULLTEXT index key

Dynamic Data Masking 3-6

SQL Server 2016

Creating a DDM

```
CREATE TABLE NewGroup
 (UID int IDENTITY PRIMARY KEY,
 FNM varchar(100) MASKED WITH (FUNCTION = 'partial(1,'XXXXXXX',0)') NULL,
 FatherName varchar(100) NOT NULL,
 Mobile varchar(12) MASKED WITH (FUNCTION = 'default()') NULL,
 PersonalEmail varchar(100) MASKED WITH (FUNCTION = 'email()') NULL);
 INSERT NewGroup (FNM, FatherName, Mobile, PersonalEmail) VALUES
('Johnson', 'Flanagan', '12345688', 'JohnnsonFlan@yahoo.com'),
('Rossell', 'Geller', '76543218', 'RossGeller@hotmail.com'),
('Brook', 'Darwin', '88956585', 'Brookdarwin@gmail.com');
SELECT * FROM NewGroup
```

Dynamic Data Masking 4-6

SQL Server 2016

Granting SELECT Permission

```
 CREATE USER DemoUser WITHOUT LOGIN;
GRANT SELECT ON NewGroup TO DemoUser;

EXECUTE AS USER = 'DemoUser';
SELECT * FROM NewGroup;
REVERT;
```

The data is changed from:

1 John Flanagan 12345688 Johnson@yahoo.com

To:

1 JXXXXXXX Flanagan xxxx JXXXX@XXXX.com

Dynamic Data Masking 5-6

SQL Server 2016

Adding or Editing a Mask on an Existing Column

Use ALTER TABLE command to

- Add a mask to the column already existing in the table
- Modify the column mask

A masking function is added to the FatherName column as shown

```
ALTER TABLE NewGroup
ALTER COLUMN FatherName ADD MASKED WITH (FUNCTION
= 'partial(2,'XXX',0)');
```

The masking function on column FatherName can be modified as

```
ALTER TABLE NewGroup
ALTER COLUMN FatherName varchar(100) MASKED WITH
(FUNCTION = 'default()');
```

Dynamic Data Masking 6-6

SQL Server 2016

Dropping a DDM

The mask on the FatherName column created earlier can be removed as shown

```
ALTER TABLE NewGroup
ALTER COLUMN FatherName DROP MASKED;
```

Row-Level Security 1-2

Row–Level security applies security rules on per row basis.

Implementation

- Using a special inline table valued function is used
- Simulating stored procedures or table valued functions
- Ensure that the rules are not bypassed

Practical Effects

- Row-Level security users will not be able to view the rows for which they do not have permission
- A Full Text Search to the column can leak the data.

Row-Level Security 2-2

To view the statistics on a secured column, the user should belong to one of the following roles:

- Table owner
- Member of sysadmin fixed server role
- db_ddladmin fixed database role
- db_owner fixed database role

Opening JSON with OPENJSON() 1-6

SQL Server 2016

JSON is an open standard that is used to send out data between Web application and a server.

- Uses text to transmit data objects in the form of attribute-value pairs.
- Used as a substitute to XML.

OPENJSON is a built-in Table-valued Function (TVF) and does the following:

- Locate an array of JSON objects
- Seeks into some JSON text
- Iterate through array elements
- Outputs one row for each array element

Opening JSON with OPENJSON() 2-6

SQL Server 2016

OPENJSON will transform the JSON text generated with FOR JSON clause back to relational form.

Two types of OPENJSON TVF are:

- **OPENJSON with predefined result**: The user can define schema for the table to be returned and the mapping rules.
- **OPENJSON without return schema**: Here the output will be as key-value pairs.

Opening JSON with OPENJSON() 3-6

SQL Server 2016

OPENXML is used as the framework for OPENJSON. Though the usage looks identical, there are elemental differences:

OPENJSON	OPENXML
Accepts text input Drawback: When two identical JSON text are passed to different OPENJSON calls, they will be parsed each time.	Accepts handle.
Return row with (key, value) schema.	Parses entire XML tree without distinct schema. Returns a flat table with all nodes.

Opening JSON with OPENJSON() 4-6

SQL Server 2016

Converting JSON to Rowset Data Using the OPENJSON Function:

The OPENJSON rowset function is used to convert the data to relational format. It returns following three values:

- **Key**: Key is either the index of an array element or property name within the object.
- **Value**: Value specifies the array element value or the value of the property within the object.
- **Type**: The data type of the value. This is depicted numerically as shown:

Numeric Value	Data Type
0	null
1	string
2	int
3	True or false
4	array
5	object

Opening JSON with OPENJSON() 5-6

SQL Server 2016

Converting JSON to Rowset Data Using the OPENJSON Function:
The user can allocate JSON snippet to a variable and then use OPENJSON function to call the variable as shown in the example:

```
DECLARE @JSON NVARCHAR (MAX) = N'

{
'InitialName': null,
'MaidenName': 'Dawson ',
'UID': 9988776655,
'Current': false,
'Skills': ['DevOps', 'Python', 'Perl'],
'Region': {'Country':'USA','Territory':'North America'}
}';

SELECT * FROM OPENJSON (@json);
```

Opening JSON with OPENJSON() 6-6

SQL Server 2016

The JSON snippet consists of a single object with a property for each data type. The SELECT statement utilizes OPENJSON function with the FROM clause to fetch JSON data as rowset. This is as shown:

Key	Value	Type
InitialName	Null	0
MaidenName	Dawson	1
UID	998776655	2
Current	False	3
Skills	['DevOps','Python','Perl']	4
Location	{'Country':'USA','Territory':'North America'}	5

Note : The type' column signifies the data type for each value. The 'key' skills is an array that consists of all array elements in the result set. The 'key' location is an object that consists of all the object properties.

.

Summary

- Security enhancements in SQL Server 2016 are Always Encrypted, Dynamic Data Mask, Transparent Data Encryption, and Row-Level security.
- Always Encrypted feature protects sensitive data .The encrypted keys are hidden from SQL database.
- Randomized encryption and deterministic encryption are the two types of encryption supported by Always Encrypted.
- The TDE technology encrypts database files, providing file level encryption. TDE protects data at rest.
- Dynamic data masking masks the data and prevents it from unauthorized access.
- Row –Level security applies security rules on per row basis.
- JSON is an open standard that is used to exchange data between Web application and a server.
- The built-in OPENJSON rowset function converts the data to relational format.