**ЛАБОРАТОРНА РОБОТА №5**

Шифрування БД

***Мета роботи:*** дослідження механізмів шифрування БД і їх впливу на швидкість операцій читання / запису.

**Хід роботи**

**Завдання 1:**

1. Створіть в базі даних симетричний ключ з алгоритмом RSA. Ключ повинен бути захищений паролем.

2. Створіть у базі даних копію таблиці. Всі дані в ній повинні бути зашифровані за допомогою створеного вами симетричного ключа.

3. Виконайте запит, який би повернув всі дані з зашифрованою таблиці. Виконати аналогічні операції з використанням асиметричних ключів і сертифікатів.

**Завдання 2.**

Зашифруйте БД за допомогою прозорого шифрування. Провести аналіз продуктивності і трасування результатів.

**Завдання 3.**

Використовуючи https://docs.microsoft.com/ru-ru/sql/relational-databases/security/encryption/encrypt-a-column-of-data створіть для таблиці зашифрований стовпець даних.

**Завдання 4.**

Використовуючи https://docs.microsoft.com/ru-ru/sql/relational-databases/security/encryption/always-encrypted-database-engine створити таблицю з зашифрованими за технологією Always Encryption даними, і https:// docs.microsoft.com/ru-ru/sql/relational-databases/security/encryption/configure-always-encrypted-using-sql-server-management-studio переглянути зашифровані і розшифровані дані.

**Шифрування за допомогою сертифікатів для захисту ключів**

Створення сертифіката, за допомогою команди CREATE CERTIFICATE. Найпростіший варіант цієї команди виглядає так:

CREATE CERTIFICATE MyCertificate

ENCRYPTION BY PASSWORD = '10210786'

WITH SUBJECT = 'Перевірка шифрування',

START\_DATE = '02.12.2018'

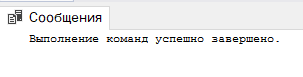


Рисунок 1 - Результат створення сертифікату

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'P@ssw0rd';

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'qwerty123QWERTY';



Рисунок 2 - Результат виконання команди

CREATE MASTER KEY ENCRIPTION

Після того, як сертифікат створений, його можна використовувати для шифрування даних. Для цієї мети застосовується спеціальна функція EncryptByCert:

CREATE TABLE TestCert (ID INT NOT NULL PRIMARY KEY, NUMBER INT NOT NULL)

INSERT INTO TestCert

VALUES (1, EncryptByCert(Cert\_ID('MyCertificate'), N'111001')),

(2, EncryptByCert(Cert\_ID('MyCertificate'), N'111002')),

(3, EncryptByCert(Cert\_ID('MyCertificate'), N'111003'))

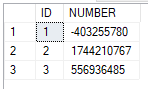


Рисунок 3 - Результат виконання шифрування

Для розшифрування використовується команда DecryptByCert. Щоб розшифруютьвать дані, потрібно використовувати звичайний запит SELECT:

SELECt CONVERT(NVARCHAR(50), DecryptByCert(Cert\_ID('Cert1'), Cred\_ID, N'11') ) FROM TestCert

**Шифрування асиметричним ключем**

Спочатку потрібно створити асиметричний ключ.

CREATE ASYMMETRIC KEY ASMKey

WITH ALGORITHM = RSA\_2048

ENCRYPTION BY PASSWORD = '10210786'



Рисунок 4 - Результат створення асиметричного ключа

Зверномо увагу, що, крім пароля, тут потрібно вказати довжину створюємого ключа. У нашому розпорядженні три варіанти: 512, 1024 і 2048 біт.

Після цього за допомогою створеного ключа можна виробляти шифрування даних:

INSERT INTO TestCert

VALUES (4, EncryptByAsymKey(AsymKey\_ID('ASMKey'), N'111001')),

(5, EncryptByAsymKey(AsymKey\_ID('ASMKey'), N'111002')),

(6, EncryptByAsymKey(AsymKey\_ID('ASMKey'), N'111003'))

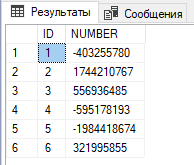


Рисунок 5 - Результат виконання асиметричного шифрування (останні три записи)

Ми вносимо в таблицю 3 записи з одним і тим же асиметричним ключем ASMKey. В результаті дані в таблиці будуть представлені у вигляді нечитаностіго набору символів. Для розшифрування скористаємося функцією DecryptByAsymKey:  
SELECT CONVERT(NVARCHAR(50), DecryptByAsymKey(AsymKey\_ID('ASMKey'), Cred\_ID, N'11')) FROM TestCert

**Шифрування симетричним ключем**

При використанні симетричних ключів шифрування проводиться швидше, ніж при застосуванні асиметричних алгоритмів, тому при роботі з біль-шими обсягами даних рекомендується використовувати саме їх. Застосування симетричних ключів виглядає дуже схоже. Правда, є і невеликі відмінності.

По-перше, при створенні симетричного ключа його можна захищати не тільки паролем, а й іншим симетричним ключем, асиметричним ключем або сертифікатом.

По-друге, при створенні симетричного ключа ви можете вказати один з восьми алгоритмів шифрування, підтримуваних SQL Server 2005 (DES, TRIPLE\_DES, RC2, RC4, DESX, AES\_128, AES\_192, AES\_256). Саме створення симетричного ключа може виглядати так:

CREATE SYMMETRIC KEY MySymKey WITH ALGORITHM = AES\_128 ENCRYPTION BY PASSWORD = '10210786'



Рисунок 6 – Результат виконаня створення симетричного ключа

Перед використанням ключа його потрібно обов'язково відкрити. Це достаточно зробити тільки один раз протягом сеансу роботи користувача:

OPEN SYMMETRIC KEY MySymKey DECRYPTION BY PASSWORD = '10210786'



Рисунок 7 - Відриття симетричного ключа

Використаєм створений ключ для шифрування даних:

INSERT INTO TestCert

VALUES (7, EncryptByKey(Key\_GUID('MySymKey'), N'111001')),

(8, EncryptByKey(Key\_GUID('MySymKey'), N'111002')),

(9, EncryptByKey(Key\_GUID('MySymKey'), N'111003'))

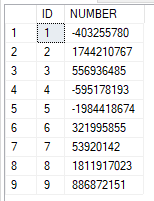


Рисунок 8 – Шифрування даних (останні три записи)   
за допомогою симетричного ключа

Звертаємо увагу, що при розшифровці даних немає необхідності передавати функції DecryptByKey ім'я симетричного ключа і пароль. Будуть автоматино підставлятися дані відкритого ключа за допомогою команди open. Для розшифровки повідомлення потрібно запустити наступний скрипт:

SELECT CONVERT(NVARCHAR(50), DecryptByKey(Cred\_ID)) FROM TestCert

Тому як зашифровані дані можна зберігати в шпальтах типу int, char, пріменяетcя команда convert.

**Прозоре шифрування**

Порядок включення шифрування:

1. Створити головний ключ;

2. Створити або отримати сертифікат, захищений головним ключем;

3. Створити ключ шифрування бази даних і захистити його за допомогою сертифіката;

4. Поставити ведення шифрування бази даних.

**Перейдем к самой реализации:**

--1) Создание главного ключа шифрования

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'StrongPassword#1';

--2) Созданный наш ключ можно увидеть в view:

select \* from sys.key\_encryptions

--3) Удалить ключ можно инструкцией:

drop master key

-- После того как создали главный ключ, необходимо сделать его резервную копию и

-- поместить резервную копию в надежное место:

BACKUP MASTER KEY TO FILE = 'D:\masterkey\_backup.bak' ENCRYPTION BY PASSWORD = '10210786'

--4) Создание сертификата

CREATE CERTIFICATE TDECertificate WITH SUBJECT ='TDE Certificate for DBClients'

--5) Проверка наличия созданного сертификата:

select \* from sys.certificates where name='TDECertificate'

--6) Создание резервной копии сертификата с закрытым ключом:

BACKUP CERTIFICATE TDECertificate

TO FILE = 'D:\cert\_TDECertificate'

WITH PRIVATE KEY

(

FILE = 'D:\sqltest2012SQLPrivateKeyFile',

ENCRYPTION BY PASSWORD = 'Password#3'

);

--7) Создание ключа шифрования в нашей базе данных с использование нашего сертификата

CREATE DATABASE ENCRYPTION KEY

WITH ALGORITHM = AES\_128

ENCRYPTION BY SERVER CERTIFICATE TDECertificate;

--8) И наконец включаем шифрование для нашей базы данных

ALTER DATABASE [DBClients]

SET ENCRYPTION ON ;

--9) В итоге имеем базу данных с прозрачным шифрованием.

**Теоретична інформація**

Проверить, что Database Encryption Key (DEK) действительно создан, можно с помощью системного представления sys.dm\_database\_encryption\_keys.

|  |
| --- |
| SELECT DB\_NAME(database\_id), \* FROM sys.dm\_database\_encryption\_keys |

https://rsdn.org/article/db/liberman_files/image004.png

В этот момент все готово для того, чтобы включить шифрование базы данных. Включаем.

|  |
| --- |
| ALTER DATABASE MySecretDB SET ENCRYPTION ON |

С этого момента начинается процесс первоначального шифрования базы данных. Он выполняется "в фоне" в отдельном потоке. Отследить прогресс выполнения этой операции можно по столбцу percent\_complete уже упомянутого нами ранее системного представления sys.dm\_database\_encryption\_keys. Так, если выполнить приведенный ниже запрос в процессе выполнения первоначального шифрования базы данных, то мы можем получить, например, следующий результат:

|  |
| --- |
| SELECT DB\_NAME(database\_id), encryption\_state, percent\_complete FROM sys.dm\_database\_encryption\_keys |

https://rsdn.org/article/db/liberman_files/image006.gif

А когда процесс первоначального шифрования базы данных будет завершен, запрос вернет следующий результат:

https://rsdn.org/article/db/liberman_files/image007.gif

В столбце encryption\_state содержится информация о текущем состоянии базы данных. Согласно SQL Server Books Online (BOL), в контексте Transparent Data Encryption (TDE) БД может находиться в одном из следующих состояний:

* 0 - Database Encryption Key (DEK) не создан.
* 1 - Database Encryption Key (DEK) создан, но база данных не зашифрована.
* 2 - Выполняется первоначальное шифрование.
* 3 - База данных зашифрована.
* 4 - Идет смена ключа.
* 5 - Идет расшифровка.

Я думаю, что вы уже обратили внимание на то, что в результате включения шифрования для нашей БД база данных tempdb также стала шифроваться. Тому, что именно шифруется для обеспечения безопасности данных, посвящен следующий раздел.

**Что именно шифруется?**

Когда для БД включено Transparent Data Encryption (TDE), шифруются как ее файлы данных, так и ее журнал транзакций.

Кроме того, как только на экземпляре SQL Server включается шифрование хотя бы одной БД, база данных tempdb также начинает шифроваться. За что "пострадала" база данных tempdb, понятно, – она может содержать куски секретной информации из шифруемых баз. А вот за что должны "страдать" приложения, работающие с другими, не зашифрованными базами данных? Их запросы, выполнение которых требует участия базы данных tempdb (большие сортировки, например), очевидно, станут выполняться медленнее. Дело, видимо, в том, что не всегда возможно определить источник данных, которые попадают в tempdb, и поэтому для гарантии она шифруется целиком.

**Файлы данных**

Когда для базы данных включается шифрование, SQL Server, как уже упоминалось выше, в отдельном потоке выполняет шифрование всех файлов данных этой БД. Но есть области, которые остаются незашифрованными:

* File Header Page (Page \*:0). Это первая страница, которая присутствует во всех файлах данных.
* Boot Page (Page 1:9) . Эта страница присутствует только в первом файле данных БД и расположена по смещению 0x12000 байт от начала файла. Именно здесь хранится зашифрованный сертификатом Database Encryption Key (DEK), а так же почти вся информация, которая доступна через системное представление sys.dm\_database\_encryption\_keys. Для отображения содержимого этой страницы, помимо универсальной команды DBCC PAGE, предназначена команда DBCC DBINFO (информацию, относящуюся к TDE она, к сожалению, не возвращает).
* Заголовки страниц (первые 0x60 байт каждой страницы) также остаются незашифрованными.

Когда SQL Server зашифровывает страницу, он устанавливает для нее соответствующий флаг (в поле m\_flagBits, которое физически расположено по смещению 4 от начала страницы и занимает 2 байта, устанавливается бит 0x800). Интересно, что мы никогда не увидим этот бит установленным через DBCC PAGE, так как в памяти все страницы расшифрованы (хотя в файле флаг для этой страницы может быть установлен).

**Журнал транзакций**

В отличие от файлов данных, операция первоначальной шифровки для журналов транзакций не выполняется. То есть информация о транзакциях, которая уже есть в журналах транзакций на момент включения шифрования, остается незащищенной. Шифруется только информация о новых транзакциях. По крайней мере, такое поведение мы можем наблюдать в CTP6.

Как следствие, если есть полная резервная копия БД до того как она была зашифрована, то даже после включения шифрования, можно сделать резервную копию журнала транзакций уже зашифрованной БД, а потом восстановить ее на момент, предшествующий шифрованию, и получить доступ к секретным данным. При этом восстановить такой архив можно без доступа к ключу (например, на другом сервере).

Следующий сценарий демонстрирует эту возможность:

|  |
| --- |
| USE master  go  -- Создаем главный ключ базы данных master  IF(not EXISTS(SELECT \* FROM sys.symmetric\_keys WHERE name = '##MS\_DatabaseMasterKey##'))  CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'My$Strong$Password$123'  go  -- Создаем сертификат, которым будем шифровать DEK  CREATE CERTIFICATE DEK\_EncCert WITH SUBJECT = 'DEK Encryption Certificate'  go  -- Создаем базу данных, которую будем шифровать  CREATE DATABASE MySecretDB  go  -- И сразу делаем ее полную резервную копию (секретных данных здесь нет)  BACKUP DATABASE MySecretDB TO DISK = 'c:\temp\MySecretDB.bak' WITH INIT  go  USE MySecretDB  go  -- Создаем таблицу и заполняем ее секретными данными  -- Делаем это в транзакции с меткой T1  BEGIN TRAN T1 WITH MARK  CREATE TABLE dbo.MySecretTable (Data varchar(200) not null)  INSERT dbo.MySecretTable (Data) VALUES ('It is my secret')  COMMIT  go  -- Шифруем базу данных  CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES\_256  ENCRYPTION BY SERVER CERTIFICATE DEK\_EncCert  go  ALTER DATABASE MySecretDB SET ENCRYPTION ON  go |

Проверяем, что база данных зашифрована:

|  |
| --- |
| SELECT DB\_NAME(database\_id), encryption\_state FROM sys.dm\_database\_encryption\_keys |

https://rsdn.org/article/db/liberman_files/image008.gif

Делаем резервную копию журнала транзакций (база данных уже зашифрована):

|  |
| --- |
| BACKUP LOG MySecretDB TO DISK = 'd:\temp\MySecretDB.trn' |

Стираем нашу базу данных, а затем и сертификат, которым мы шифровали ее Database Encryption Key (DEK). Нам это нужно для того, чтобы эмулировать восстановление БД на другом сервере.

|  |
| --- |
| USE master  go  DROP DATABASE MySecretDB  go  DROP CERTIFICATE DEK\_EncCert  go |

Теперь попытаемся восстановить базу данных. Сначала полностью:

|  |
| --- |
| RESTORE DATABASE MySecretDB FROM DISK = 'd:\temp\MySecretDB.bak' WITH NORECOVERY  RESTORE LOG MySecretDB FROM DISK = 'd:\temp\MySecretDB.trn' |

Как и следовало ожидать, попытка восстановления базы данных закончилась ошибкой. Сертификат, которым зашифрован Database Encryption Key (DEK), более недоступен.

|  |
| --- |
| Msg 33111, Level 16, State 3, Line 2  Cannot find server certificate with thumbprint '0x347D263A185EF41D8EB06AE425F7599AD2D0FCC3'.  Msg 3013, Level 16, State 1, Line 2  RESTORE LOG is terminating abnormally. |

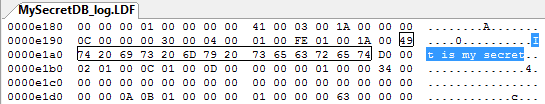
А теперь восстановим базу данных на момент "до включения шифрования", то есть на отметку T1.

|  |
| --- |
| RESTORE DATABASE MySecretDB FROM DISK = 'd:\temp\MySecretDB.bak' WITH NORECOVERY  RESTORE LOG MySecretDB FROM DISK = 'd:\temp\MySecretDB.trn' WITH STOPATMARK = 'T1'  go  USE MySecretDB  go  -- Запрос к секретным данным  SELECT \* FROM dbo.MySecretTable |

Все, доступ к секретным данным мы получили:

https://rsdn.org/article/db/liberman_files/image009.gif

Напомню, что сами секретные данные были сброшены в резервную копию журнала транзакций уже после того, как база данных была зашифрована и восстановлена без всякого доступа к ключу. На самом деле для того чтобы увидеть секретные данные, в нашем случае достаточно было открыть в редакторе файл журнала транзакций (или его резервную копию). Несмотря на то, что наша база зашифрована, секретные данные в журнале транзакций остались в открытом виде (шифруются только новые транзакции)



**Висновок:** за допомогою Лабораторної роботи №5 ми дослідили механізми шифрування БД і їх впливу на швидкість операцій читання / запису. Також з теоретичної інформації, ми робимо наступний висновок: якщо ми шифруємо БД, яка вже містить секретну інформацію, то ми повинні або перебудувати журнал транзакцій, або подбати про те, щоб незашифровані транзакції в журналі транзакцій були перезаписані новими зашифрованими транзакціями.