Лабораторна робота №6
СТВОРЕННЯ ПЛАНУ ОБСЛУГОВУВАННЯ

Мета роботи: дослідження завдань та інструментарію серверів БД для організації регламентованого обслуговування.

Основи коротко: http://www.flenov.info/books.php?contentid=47

Завдання

1. Створити скрипти основних процедур обслуговування БД
2. Налаштувати регламент виконання операцій та розклад для плану обслуговування, за допомогою SQL SERVER агент (або ін. планувальників завдань).

Вимоги до звіту

1. Документ зі скріншотами екрану, що містять результати виконання завдань.
2. Файл-скрипт.

Теоретичні відомості

Помилково розглядати базу даних як якусь еталонну одиницю, оскільки, з часом, можуть виявлятися різноманітні небажані ситуації – деградація продуктивності, збої у роботі та інше.

Для мінімізації ймовірності виникнення таких ситуацій створюють плани обслуговування із завдань, що гарантують стабільність та оптимальну продуктивність бази даних.

Серед таких завдань можна виділити такі:

1. Дефрагментація індексів
2. Оновлення статистики
3. Резервне копіювання

Крім фрагментації файлової системи та лог-файлу, відчутний вплив на продуктивність бази даних надає фрагментація всередині файлів даних:

1. Фрагментація всередині окремих сторінок індексу

Після операцій вставки, оновлення та видалення записів неминуче виникають порожні місця на сторінках. Нічого страшного в цьому немає, оскільки ця ситуація цілком нормальна, якби не одне але…

Дуже важливу роль відіграє довжина рядка. Наприклад, якщо рядок має розмір, який займає більше половини сторінки, вільна половина цієї сторінки не використовуватиметься. У результаті зі збільшенням кількості рядків спостерігатиметься зростання невикористовуваного місця у базі даних.

Боротися з цим видом фрагментації варто на етапі проектування схеми, тобто вибирати такі типи даних, які компактно вміщалися б на сторінках.

2. Фрагментація всередині структур індексу

Основною причиною виникнення цього виду фрагментації є операції розбиття сторінки. Наприклад, згідно з структурою первинного ключа, новий рядок необхідно вставити на певну сторінку індексу, але цієї на сторінці недостатньо місця, щоб розмістити дані, що вставляються.

У такому разі створюється нова сторінка, на яку переміститься приблизно половина записів зі старої сторінки. Нова сторінка часто не є фізично суміжною зі старою і, отже, позначається системою як фрагментована.

У будь-якому випадку, фрагментація веде до зростання кількості сторінок для зберігання того ж обсягу інформації. Це автоматично призводить до збільшення розміру бази даних та зростання місця, що не використовується.

При виконанні запитів, в яких йде звернення до фрагментованих індексів, потрібно більше операцій IO . Крім того, фрагментація накладає додаткові витрати на пам'ять сервера, якому доводиться зберігати в кеші зайві сторінки.

Для боротьби з фрагментацією індексів у арсеналі *SQL Server* передбачені команди: *ALTER INDEX REBUILD / REORGANIZE* .

Перебудова індексу передбачає видалення старого та створення нового екземпляра індексу. Ця операція усуває фрагментацію, відновлює дисковий простір шляхом ущільнення сторінки, резервуючи вільне місце на сторінці, яке можна задати опцією *FILLFACTOR* . Важливо відзначити, що операція з перебудови індексу дуже затратна .

Тому, якщо фрагментація незначна, переважно реорганізовувати існуючий індекс. Ця операція вимагає менших системних ресурсів, ніж перестворення індексу і полягає у реорганізації *leaf-level* сторінок. Крім того , реорганізація при можливості стискає сторінки індексів.

Власне навіщо потрібна статистика?

При виконанні будь-якого запиту, оптимізатор запитів, в рамках наявної в нього інформації, намагається побудувати оптимальний план виконання - який відображатиме послідовність операцій, за рахунок виконання яких можна отримати необхідний результат, описаний у запиті.

У процесі вибору тієї чи іншої операції, оптимізатор запитів до найважливіших вхідних даних відносить статистику, що описує розподіл значень даних для стовпців всередині таблиці або індексу.

Така оцінка кількості елементів дозволяє оптимізатору запитів створювати ефективніші плани виконання. У той самий час, якщо статистика міститиме застарілі дані, може бути обрано менш ефективні операції, що призведе до створення повільних планів виконання. Наприклад, коли для невеликої вибірки на застарілій статистиці вибирається більш витратний оператор Index Scan замість оператора Index Seek .

Як Ви бачите, щоб бути максимально корисною для оптимізатора запитів, статистика має бути точною та свіжою. Іноді SQL Server періодично сам оновлює статистику - ця поведінка регулюється опціями AUTO_CREATE_STATISTICS та AUTO_UPDATE_STATISTICS .

Крім того, при перестворенні індексів статистика по ним оновлюється автоматично з увімкненим прапором FULLSCAN , що гарантує найточніше розподіл даних. При реорганізації індексів статистика не оновлюється.

Коли дані в таблицях дуже часто змінюються, доцільно виконувати виборче оновлення статистики вручну, за допомогою операції UPDATE STATISTICS .

Також ручне оновлення дуже важливо, коли для статистики заданий прапор NORECOMPUTE , що означає, що автоматичне оновлення статистики в подальшому не потрібно.

Існує велика кількість постів, у яких наполегливо закликають до однієї простої істини – потрібно робити бекапи на постійній основі.

Хід роботи

1. Автоматична дефрагментація індексів

Ступінь фрагментації того чи іншого індексу можна дізнатися з динамічного системного представлення **sys.dm_db_index_physical_stats** :

```
SELECT *
FROM sys.dm_db_index_physical_stats ( DB_ ID ( ), NULL, NULL, NULL, NULL)
WHERE avg_fragmentation_in_percent > 0
```

В даному запиті останній параметр задає режим, від значення якого можливо швидке, але не зовсім точне визначення рівня фрагментації індексу (режими *LIMITED / NULL*). Тому рекомендується задавати режими *SAMPLED / DETAILED* .

Ми знаємо, звідки отримати список фрагментованих індексів. Тепер необхідно для кожного з них згенерувати відповідну *ALTER INDEX* команди. Традиційно для цього використовують курсор :

```
DECLARE @SQL NVARCHAR(MAX)
DECLARE cur CURSOR LOCAL READ_ONLY FORWARD_ONLY FOR
SELECT '
ALTER INDEX [' + i.name + N'] ON [' + SCHEMA_NAME(o.[schema_id]) + '].[' + o.name
+ '] ' + CASE WHEN s.avg_fragmentation_in_percent > 30
THEN 'REBUILD WITH (SORT_IN_TEMPDB = ON)'
ELSE 'REORGANIZE'
END + ';'
FROM (
SELECT
s.[object_id]
, s.index_id
, avg_fragmentation_in_percent = MAX(s.avg_fragmentation_in_percent)
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, NULL) s
WHERE s.page_count > 128 -- > 1 MB
AND s.index_id > 0 -- <> HEAP
AND s.avg_fragmentation_in_percent > 5
GROUP BY s.[object_id], s.index_id
) s

JOIN sys.indexes i WITH(NOLOCK) ON s.[object_id] = i.[object_id] AND s.index_id = i.in-
dex_id

JOIN sys.objects o WITH(NOLOCK) ON o.[object_id] = s.[object_id]
OPEN cur
FETCH NEXT FROM cur INTO @SQL
```

```
WHILE @@FETCH_STATUS = 0 BEGIN
EXEC sys.sp_executesql @SQL
FETCH NEXT FROM cur INTO @SQL
END
CLOSE cur
DEALLOCATE cur
```

Щоб прискорити процес перестворення індексу, рекомендується додатково вказувати опцію SORT_IN_TEMPDB. Ще потрібно окремо згадати про опцію ONLINE - вона уповільнює перестворення індексу. Але іноді буває корисним. Наприклад, читання із кластерного індексу дуже дороге. Ми створили покриваючий індекс та вирішили проблему з продуктивністю. Далі ми робимо *REBUILD* некластерного індексу. У цей момент нам доведеться знову звертатися до кластерного індексу, що знижує перфоманс.

SORT_IN_TEMPDB дозволяє перебудовувати індекси в базі *tempdb* , що особливо корисно для великих індексів у разі нестачі пам'яті і в іншому випадку - опція ігнорується. Крім того, якщо база *tempdb* розташована на іншому диску, це істотно скоротить час створення індексу. *ONLINE* дозволяє перестворити індекс , не блокуючи при цьому запити до об'єкта для якого цей індекс створюється.

Як показала практика, дефрагментування індексів з низьким ступенем фрагментації або з невеликою кількістю сторінок не приносить будь-яких помітних покращень, що сприяють підвищенню продуктивності при роботі з ними.

У додатку, наведений вище запит, можна переписати без застосування курсору:

```
DECLARE
@IsDetailedScan BIT = 0
, @IsOnline BIT = 0
DECLARE @SQL NVARCHAR (MAX)
SELECT @SQL = (
        SELECT '
        ALTER INDEX ['+i.name+N'] ON['+ SCHEMA_NAME (o.[ schema_id ])+' ].[ '+o.name+']'+
                CASE WHEN s.avg_fragmentation_in_percent > 30
                        THEN 'REBUILD WITH ( SORT_IN_TEMPDB = ON'
                                -- Enterprise, Developer
                                + CASE WHEN SERVERPROPERTY (' EditionID ') IN (1804890536, -2117995310) AND @ IsOnline = 1
                                                        THEN', ONLINE = ON'
                                                        ELSE ''
                                END + ')'
                        ELSE 'REORGANIZE'
                END+';
        '
        FROM (
                SELECT
                        s.[ object_id ]
                        , s.index _id
```

```
                , avg_fragmentation_in_percent = MAX( s.avg_fragmentation_in_percent )
                FROM sys.dm_db_index_physical_stats ( DB_ ID ( ), NULL, NULL, NULL,
                                                    CASE WHEN@ IsDetailedScan = 1
                                                        THEN 'DETAILED'
                                                        ELSE 'LIMITED'
                                                    END ) s
                WHERE s.page _count > 128 -- > 1 MB
                    AND s.index _id > 0 -- <> HEAP
                    AND s.avg_fragmentation_in_percent > 5
                GROUP BY s.[ object_id ], s.index _id
            ) s
            JOIN sys.indexes i ON s.[ object_id ] = i .[ object_id ] AND s.index_id = i.index_id
            JOIN sys.objects o ON o.[ object_id ] = s.[ object_id ]
            FOR XML PATH( ''), TYPE).value('.', ' NVARCHAR (MAX)')
    PRINT @ SQL
    EXEC sys.sp _executesql @ SQL
```

В результаті обидва запити при виконанні будуть генерувати запити щодо дефрагментації проблемних індексів:

```
ALTER INDEX [ IX_TransactionHistory_ProductID ]
ON [Production ]. [ TransactionHistory ] REORGANIZE;

ALTER INDEX [ IX_TransactionHistory_ReferenceOrderID_ReferenceOrderLineID ]
ON [Production ]. [ TransactionHistory ] REBUILD WITH ( SORT_IN_TEMPDB = ON, ONLINE = ON);

ALTER INDEX [ IX_TransactionHistoryArchive_ProductID ]
  ON [ Production ]. [ TransactionHistoryArchive ] REORGANIZE ;
```

Власне, на цьому першу частину створення плану обслуговування для бази даних виконано.

Можливість дефрагментації окремих секцій:

```
USE ...
DECLARE
@ PageCount INT = 128
, @RebuildPercent INT = 30
, @ReorganizePercent INT = 10
, @IsOnlineRebuild BIT = 0
, @ IsVersion2012Plus BIT =
```

```
	CASE WHEN CAST( SERVERPROPERTY (' productversion ') AS CHAR( 2)) NOT IN
('8.', '9.', '10')
	THEN 1
	ELSE 0
	END
	, @ IsEntEdition BIT =
	CASE WHEN SERVERPROPERTY (' EditionID ') IN (1804890536, -2117995310)
	THEN 1
	ELSE 0
	END
	, @SQL NVARCHAR (MAX)
	SELECT @SQL = (
	SELECT
	'
	ALTER INDEX '+ QUOTENAME (i.name) + 'ON' + QUOTENAME (s2.name) + '.' +
QUOTENAME (o.name) + " +
	CASE WHEN s.avg_fragmentation_in_percent >= @RebuildPercent
	THEN 'REBUILD'
	ELSE 'REORGANIZE'
	END + 'PARTITION=' +
	CASE WHEN ds.[ type] != 'PS'
	THEN 'ALL'
	ELSE CAST( s.partition_number AS NVARCHAR (10))
	END + 'WITH ('+
	CASE WHEN s.avg_fragmentation_in_percent >= @RebuildPercent
	THEN ' SORT_IN_TEMPDB =ON' +
	CASE WHEN @ IsEntEdition = 1
	AND @ IsOnlineRebuild = 1
	AND ISNULL ( lob.is_lob_legacy , 0) = 0
	AND (
				ISNULL ( lob.is_lob , 0) = 0
	OR
	( lob.is_lob = 1 AND @ IsVersion2012Plus = 1)
	)
	THEN', ONLINE = ON'
	ELSE "
	END
	ELSE ' LOB_COMPACTION =ON'
	END + ')'
	FROM sys.dm_db_index_physical_stats ( DB_ ID ( ), NULL, NULL, NULL, NULL) s
	JOIN sys.indexes i ON i .[ object_id ] = s.[ object_id ] AND i.index_id = s.index_id
	LEFT JOIN (
	SELECT
	c.[ object_id ]
	, index_id = ISNULL ( i.index_id , 1)
	, is_lob_legacy = MAX( CASE WHEN c.system_type_id IN (34, 35, 99) THEN 1 END)
```

```
, is_lob = MAX( CASE WHEN c.max_length = -1 THEN 1 END)
FROM sys.columns c
LEFT JOIN sys.index _columns i ON c.[ object_id ] = i .[ object_id ]
AND c.column _id = i.column_id AND i.index_id > 0
WHERE c.system _type_id IN (34, 35, 99)
        OR c.max_length = -1
    GROUP BY c.[ object_id ], i.index _id
) lob ON lob.[ object_id ] = i .[ object_id ] AND lob.index_id = i.index_id
JOIN sys.objects o ON o.[ object_id ] = i .[ object_id ]
JOIN sys.schemas s2 ON o.[ schema_id ] = s2 .[ schema_id ]
JOIN sys.data_spaces ds ON i.data_space_id = ds.data_space_id
WHERE i .[type] IN (1, 2)
AND i.is_disabled = 0
    AND i.is_hypothetical = 0
    AND s.index _level = 0
AND s.page _count > @ PageCount
AND s.alloc _unit_type_desc = ' IN_ROW_DATA '
AND o.[type] IN ('U', 'V')
AND s.avg_fragmentation_in_percent > @ ReorganizePercent
FOR XML PATH( ''), TYPE
). value( '.', ' NVARCHAR (MAX)')
PRINT @SQL
--EXEC sys.sp _executesql @SQL
```

2. Оновлення статистики

Переглянути цю властивість, як і на всі інші, можна у властивостях статистики:

```
SELECT s.*
FROM sys.stats s
JOIN sys.objects o ON s.[ object_id ] = o.[ object_id ]
WHERE o.is_ms_shipped = 0
```

Застосовуючи можливості динамічного SQL , напишемо скрипт з автоматичного оновлення застарілої статистики:

```
DECLARE @DateNow DATETIME
SELECT @DateNow = DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()))
DECLARE @SQL1 NVARCHAR(MAX)
SELECT @SQL1 = (
SELECT '
UPDATE STATISTICS ['+ SCHEMA_NAME(o.[schema_id]) + '].['+ o.name
+'] [' +
s.name + ']
WITH FULLSCAN' + CASE WHEN s.no_recompute = 1 THEN ', NORECOM-
PUTE' ELSE '' END + ';'
FROM sys.stats s WITH(NOLOCK)
JOIN sys.objects o WITH(NOLOCK) ON s.[object_id] = o.[object_id]
```

```
WHERE o.[type] IN ('U', 'V')
AND o.is_ms_shipped = 0
AND ISNULL(STATS_DATE(s.[object_id], s.stats_id), GETDATE()) <=
@DateNow
FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)')
PRINT 'DB statistics update'
EXEC sys.sp_executesql @SQL1
```

При виконанні будуть генеруватися наступні стейтменти :

UPDATE STATISTICS [Production].[Shift] [PK_Shift_ShiftID] WITH FULLSCAN ;

UPDATE STATISTICS [Production].[Shift] [AK_Shift_Name] WITH FULLSCAN , NORECOMPUTE ;

Критерій старіння статистики у кожній конкретній ситуації може бути свій. У цьому прикладі – 1 день.

У деяких випадках, надто приватне оновлення статистики для великих таблиць може помітно знижувати продуктивність бази даних, тому цей скрипт можна модифікувати. Наприклад, для великих таблиць оновлювати статистику рідше:

```
DECLARE @DateNow DATETIME
SELECT @DateNow = DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()))
DECLARE @SQL NVARCHAR(MAX)
SELECT @SQL = (
SELECT '
UPDATE STATISTICS [' + SCHEMA_NAME(o.[schema_id]) + '].[' + o.name + '] [' + s.name +
']
WITH FULLSCAN' + CASE WHEN s.no_recompute = 1 THEN ', NORECOMPUTE' ELSE '' END + ';'
FROM (
   SELECT
    [object_id]
   , name
   , stats_id
   , no_recompute
   , last_update = STATS_DATE([object_id], stats_id)
   FROM sys.stats WITH(NOLOCK)
   WHERE auto_created = 0
AND is_temporary = 0 -- 2012+
s
JOIN sys.objects o WITH(NOLOCK) ON s.[object_id] = o.[object_id]
JOIN (
SELECT
p.[object_id], p.index_id, total_pages = SUM(a.total_pages) FROM sys.partitions p
WITH(NOLOCK)
JOIN sys.allocation_units a WITH(NOLOCK) ON p.[partition_id] = a.container_id
```

```
    GROUP BY
    p.[object_id]
    , p.index_id
) p ON o.[object_id] = p.[object_id] AND p.index_id = s.stats_id
       WHERE o.[type] IN ('U', 'V')
    AND o.is_ms_shipped = 0
    AND (
    last_update IS NULL AND p.total_pages > 0 -- never updated and contains rows
    OR
                              last_update <= DATEADD(dd,
                                   CASE WHEN p.total_pages > 4096 -- > 4 MB
                                          THEN -2 -- updated 3 days ago
                                          ELSE 0
                                   END, @DateNow)
                        )
            FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)')
       PRINT @SQL
       EXEC sys.sp_executesql @SQL
```

3. Автоматичне створення бекапів

Створимо план обслуговування резервного копіювання, а потім обговоримо деякі тонкощі пов'язані з бетонами .

Створимо таблицю, в якій записуватимуться повідомлення про помилки під час створення резервних копій:

```
USE [master]
GO
IF OBJECT_ ID ( ' dbo.BackupError ', 'U') IS NOT NULL
DROP TABLE dbo.BackupError
GO
CREATE TABLE dbo.BackupError (
   db SYSNAME PRIMARY KEY,
   dt DATETIME NOT NULL DEFAULT GETDATE ( ),
   msg NVARCHAR ( 2048)
)
GO
```

Скрипт для резервного копіювання баз даних на кожен день я використовую такий:

```
USE [master]
GO
SET NOCOUNT ON
TRUNCATE TABLE dbo.BackupError
DECLARE
      @db SYSNAME
    , @sql NVARCHAR(MAX)
    , @can_compress BIT
```

```
        , @path NVARCHAR(4000)
        , @name SYSNAME
        , @include_time BIT
    SET @path = 'D:\backup'
    IF @path IS NULL
        EXEC [master].dbo.xp_instance_regread
                N'HKEY_LOCAL_MACHINE',
                N'Software\Microsoft\MSSQLServer\MSSQLServer',
                N'BackupDirectory', @path OUTPUT, 'no_output'
    SET @can_compress = ISNULL(CAST((
        SELECT value
        FROM sys.configurations
        WHERE name = 'backup compression default') AS BIT), 0)
    DECLARE cur CURSOR FAST_FORWARD READ_ONLY LOCAL FOR
        SELECT d.name
        FROM sys.databases d
        WHERE d.[state] = 0
            AND d.name NOT IN ('tempdb')
    OPEN cur
    FETCH NEXT FROM cur INTO @db
    WHILE @@FETCH_STATUS = 0 BEGIN
        IF DB_ID(@db) IS NULL BEGIN
            INSERT INTO dbo.BackupError (db, msg) VALUES (@db, 'db is missing')
        END
        ELSE IF DATABASEPROPERTYEX(@db, 'Status') != 'ONLINE' BEGIN
            INSERT INTO dbo.BackupError (db, msg) VALUES (@db, 'db state != ONLINE')
        END
        ELSE BEGIN
            BEGIN TRY
                SET @name = @path + '\T' + CONVERT(CHAR(8), GETDATE(), 112) + '_' + @db +
'.bak'
                SET @sql = '
                    BACKUP DATABASE ' + QUOTENAME(@db) + '
                    TO DISK = ''' + @name + ''' WITH NOFORMAT, INIT' +
                    CASE WHEN @can_compress = 1 THEN ', COMPRESSION' ELSE '' END
                --PRINT @sql
                EXEC sys.sp_executesql @sql
            END TRY
            BEGIN CATCH
                INSERT INTO dbo.BackupError (db, msg) VALUES (@db, ERROR_MESSAGE())
            END CATCH
        END
        FETCH NEXT FROM cur INTO @db
```

```
END
CLOSE cur
DEALLOCATE cur
```

Якщо на сервері настроєно компонент *Database Mail* , то в скрипт можна додати повідомлення поштою про проблеми:

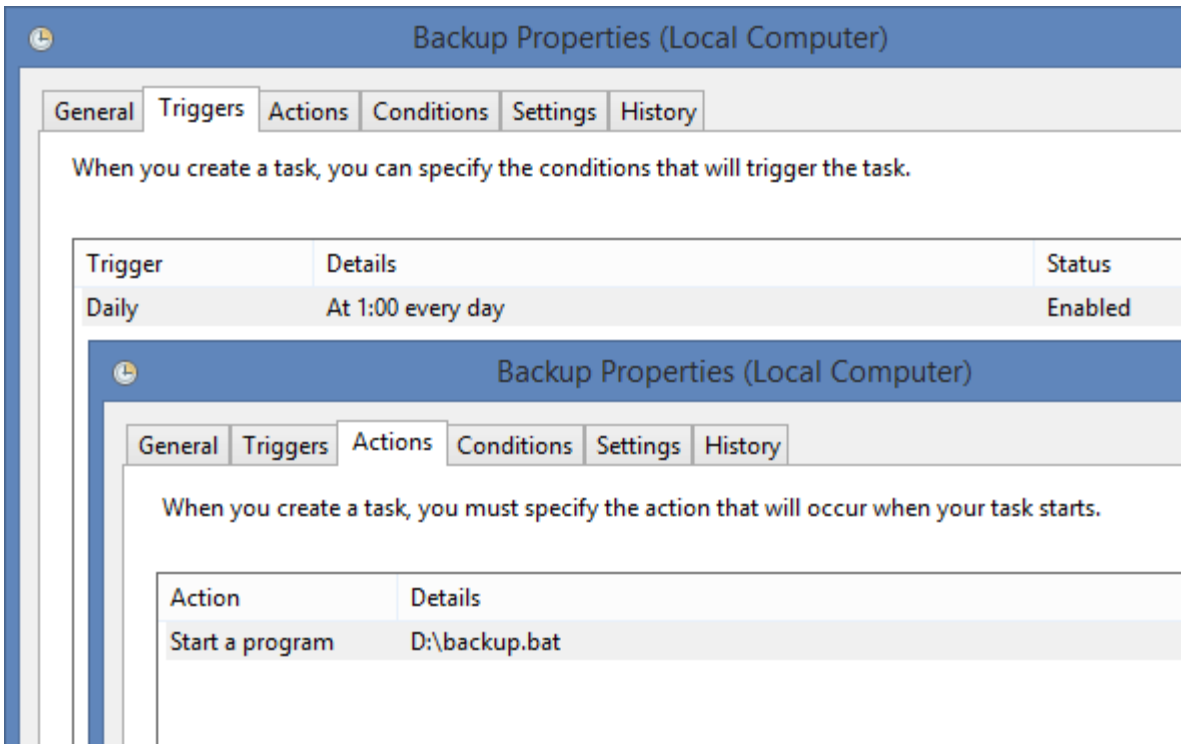
```
IF EXISTS( SELECT 1 FROM dbo.BackupError ) BEGIN
DECLARE @report NVARCHAR (MAX)
SET @ report =
'<table border="1"><tr><th>database</th><th>date</th><th>message</th></tr>' +
      CAST ( (
SELECT td = db , '', td = dt , '', td = msg
FROM dbo.BackupError
FOR XML PATH(' tr '), TYPE
) AS NVARCHAR (MAX)) +
'</table>'
EXEC msdb.dbo.sp _send_dbmail
@recipients = 'your_account@mail.ru',
@subject = 'Backup Problems',
@body = @report,
      @body_format = ' HTML '
END
```

Власне, на цьому етапі робочий скрипт для автоматичного створення резервних копій готовий. Залишається його створити *job*, який би за розкладом запускав цей скрипт.

Власників *Express* редакцій потрібно окремо згадати, оскільки *SQL Server Express edition* немає можливості використовувати *SQL Server Agent*. Який би сум не прийшов після цих слів, насправді все вирішуване. Найпростіше створити *bat* файл із приблизно схожим змістом:

```
sqlcmd - S < ComputerName >\< InstanceName > - i D :\ backup . sql
```

Далі відкрити *Task Scheduler* та створити в ньому нове завдання.



Друга альтернатива – використовувати сторонні розробки, які дозволяють запускати завдання розкладу. Серед можна виділити *SQL Scheduler* - зручний та безкоштовний тул . Інсталятор у мене загубився, тож буду вдячний, якщо хтось поділиться робочим посиланням для читачів.

Тепер поговоримо про корисні дрібниці пов'язані з бекапами .

Стиснення…

Можливість стиснення бекапів з'явилася вперше у *SQL Server 2008* . Згадую з ностальгією час, коли працюючи на *2005* версії мені доводилося *7Zip* стискати бекапи . Тепер все стало набагато простіше.

Але слід пам'ятати, що стиснення бекапів буде використовуватися лише якщо виконувати команду *BACKUP* з параметром *COMPRESSION* або увімкнути стандартне стиснення наступною командою:

```
USE [master]
GO
EXEC sp_configure 'backup compression default', 1
RECONFIGURE WITH OVERRIDE
GO
```

До слова буде сказано, що стислі бекапи має деякі переваги: потрібно менше місця для їх зберігання, відновлення БД зі стислих бекапів зазвичай виконується трохи швидше, також вони швидше створюються, оскільки вимагають меншої кількості *I/O* операцій. Мінуси, до речі, теж є – під час роботи зі стислими бекапами навантаження на процесор збільшується.

Цим запитом можна повернути розмір останнього *FULL* бекапа зі стиском і без:

```
SELECT
      database_name
    , backup_size_mb = backup_size / 1048576.0
```

```
        , compressed_backup_size_mb = compressed_backup_size / 1048576.0
        , compress_ratio_percent = 100 - compressed_backup_size * 100. / backup_size
    FROM (
       SELECT
             database_name
           , backup_size
           , compressed_backup_size = NULLIF(compressed_backup_size, backup_size)
           ,  RowNumber  =  ROW_NUMBER()  OVER  (PARTITION  BY  database_name  ORDER  BY
backup_finish_date DESC)
        FROM msdb.dbo.backupset
        WHERE [type] = 'D'
    ) t
    WHERE t.RowNumber = 1
```

Зазвичай стиск досягає 40-90%, якщо не брати до уваги бінарні дані.

Якщо модифікувати попередній запит, можна моніторити для яких баз робилися резервні копії:

```
    SELECT
       d.name
        , rec_model = d.recovery_model_desc
        , f.full_time
        , f.full_last_date
        , f.full_size
        , f.log_time
        , f.log_last_date
        , f.log_size
    FROM sys.databases d
    LEFT JOIN (
        SELECT
           database_name
         ,   full_time   =   MAX(CASE   WHEN   [type]   =   'D'   THEN   CONVERT(CHAR(10),
backup_finish_date - backup_start_date, 108) END)
         , full_last_date = MAX(CASE WHEN [type] = 'D' THEN backup_finish_date END)
         , full_size = MAX(CASE WHEN [type] = 'D' THEN backup_size END)
         ,   log_time   =   MAX(CASE   WHEN   [type]   =   'L'   THEN   CONVERT(CHAR(10),
backup_finish_date - backup_start_date, 108) END)
         , log_last_date = MAX(CASE WHEN [type] = 'L' THEN backup_finish_date END)
         , log_size = MAX(CASE WHEN [type] = 'L' THEN backup_size END)
        FROM (
        SELECT
              s.database_name
           , s.[type]
           , s.backup_start_date
           , s.backup_finish_date
```

```
            , backup_size =
                        CASE WHEN s.backup_size = s.compressed_backup_size
                                    THEN s.backup_size
                                    ELSE s.compressed_backup_size
                        END / 1048576.0
            , RowNum = ROW_NUMBER() OVER (PARTITION BY s.database_name, s.[type] ORDER
BY s.backup_finish_date DESC)
        FROM msdb.dbo.backupset s
        WHERE s.[type] IN ('D', 'L')
        ) f
        WHERE f.RowNum = 1
        GROUP BY f.database_name
    ) f ON f.database_name = d.name
```

Якщо база стоїть модель відновлення *FULL* або *BULK_LOGGED* , то бажано час від часу робити бекап лога, щоб не прирікати сервер на муки постійного зростання *LDF* файлу. Ступінь заповнення файлу даних та лога для баз даних можна переглянути цим запитом:

```
    IF OBJECT_ID('tempdb.dbo.#space') IS NOT NULL
        DROP TABLE #space
    CREATE TABLE #space (
        database_id INT PRIMARY KEY,
        data_used_size DECIMAL(18,6),
        log_used_size DECIMAL(18,6)
    )
    DECLARE @SQL NVARCHAR(MAX)
    SELECT @SQL = STUFF((
        SELECT '
        USE [' + d.name + ']
        INSERT INTO #space (database_id, data_used_size, log_used_size)
        SELECT
           DB_ID()
        , SUM(CASE WHEN [type] = 0 THEN space_used END)
        , SUM(CASE WHEN [type] = 1 THEN space_used END)
        FROM (
        SELECT s.[type], space_used = SUM(FILEPROPERTY(s.name, ''SpaceUsed'') * 8. /
1024)
        FROM sys.database_files s
        GROUP BY s.[type]
        ) t;'
        FROM sys.databases d
        WHERE d.[state] = 0
        FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)'), 1, 2, '')
```

```
EXEC sys.sp_executesql @SQL
SELECT
   database_name = DB_NAME(t.database_id)
    , t.data_size
    , s.data_used_size
    , t.log_size
    , s.log_used_size
    , t.total_size
FROM (
    SELECT
       database_id
     , log_size = SUM(CASE WHEN [type] = 1 THEN size END) * 8. / 1024
     , data_size = SUM(CASE WHEN [type] = 0 THEN size END) * 8. / 1024
     , total_size = SUM(size) * 8. / 1024
    FROM sys.master_files
    GROUP BY database_id
) t
LEFT JOIN #space s ON t.database_id = s.database_id
```