```
WITH DirectReports(name, parent_type, type, level, sort) AS
(
   SELECT CONVERT(varchar(255),type_name), parent_type, type, 1,
          CONVERT(varchar(255),type_name)
   FROM sys.trigger_event_types
   WHERE parent_type IS NULL
   UNION ALL
   SELECT  CONVERT(varchar(255), REPLICATE ('|   ' , level) +
          e.type_name),
      e.parent_type, e.type, level + 1,
   CONVERT (varchar(255), RTRIM(sort) + '|   ' + e.type_name)
   FROM sys.trigger_event_types AS e
      INNER JOIN DirectReports AS d
      ON e.parent_type = d.type
)
SELECT parent_type, type, name
FROM DirectReports
ORDER BY sort;
```

Следует хранить служебные элементы (мониторинг, администрирование и т. д.) в их собственной базе данных. Это позволяет делать запросы централизованно, а также контролировать рост по отдельности. Для этой задачи воспользуемся базой данных с именем VlasovaAuditDB:

```
CREATE DATABASE VlasovaAuditDB;
GO
```

Чтобы все было относительно просто, давайте предположим, что нас интересуют только действия, выполняемые над хранимыми процедурами — создание, изменение, удаление. У нас уже есть набор хранимых процедур, и они находятся в заданном состоянии. Нам нужно будет зафиксировать это состояние в дополнение к любым изменениям, которые будут внесены в них с этого момента. Таким образом, мы всегда сможем вернуться в любое состояние, включая исходное.

В дополнение к данным, относящимся к действиям, выполняемым над хранимыми процедурами, мы также можем подумать о нескольких других элементах информации, которые мы хотели бы хранить о каждом событии. Например:

- имя базы данных
- имя схемы/объекта
- информация для входа
- имя хоста/IP-адрес (полезно при аутентификации SQL)

```
CREATE TABLE DDLEvents
(
   EventDate    DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
   EventType    NVARCHAR(64),
   EventDDL     NVARCHAR(MAX),
   EventXML     XML,
   DatabaseName NVARCHAR(255),
   SchemaName   NVARCHAR(255),
   ObjectName   NVARCHAR(255),
   HostName     VARCHAR(64),
   IPAddress    VARCHAR(48),
   ProgramName  NVARCHAR(255),
   LoginName    NVARCHAR(255)
);
```

Да, мы могли бы уменьшить размер таблицы и использовать [object_id] вместо имени

схемы/объекта, что также защитит нас от проблем с разрешением из-за переименований. Однако часто хранимые процедуры удаляются и создаются заново, и в этом случае система сгенерирует новый [object_id]. Я также предпочитаю использовать имя базы данных, чтобы упростить специальные запросы (и автоматизацию сценариев) к конкретным базам данных. Вы можете выбрать, на какие метаданные полагаться; лично я поменяю место на удобочитаемость и возможность написания сценариев.

Теперь, когда таблица существует, мы можем легко получить снимок наших существующих определений хранимых процедур, опустив некоторые нерелевантные данные аудита, как показано ниже (заменив «мое имя» тем, что вы хотите отобразить для начальных строк):

```
INSERT DDLEvents
(
   EventType,
   EventDDL,
   DatabaseName,
   SchemaName,
   ObjectName,
   LoginName
)
SELECT
   'CREATE_PROCEDURE',
   OBJECT_DEFINITION([object_id]),
   DB_NAME(),
   OBJECT_SCHEMA_NAME([object_id]),
   OBJECT_NAME([object_id]),
   user_name(OBJECTPROPERTY([object_id],'ownerid'))
FROM
   sys.procedures;

SELECT * from sys.all_objects
 where [name]='new_user_login';
```

Теперь мы готовы начать фиксировать фактические изменения в этих процедурах по мере их возникновения. Вы можете создать триггер DDL со следующим кодом, который будет записывать соответствующие данные в приведенную выше таблицу при внесении изменений в хранимые процедуры:

```
USE VlasovaAdmin;
GO


create TRIGGER DDLTrigger_Sample
   ON DATABASE
   FOR CREATE_PROCEDURE, ALTER_PROCEDURE, DROP_PROCEDURE
AS
BEGIN
   SET NOCOUNT ON;
   DECLARE
      @EventData XML = EVENTDATA();

   DECLARE @ip varchar(48) = CONVERT(varchar(48),
      CONNECTIONPROPERTY('client_net_address'));

   INSERT vlasovaAuditDB..DDLEvents
   (
      EventType,
      EventDDL,
      EventXML,
      DatabaseName,
      SchemaName,
```

```
      ObjectName,
      HostName,
      IPAddress,
      ProgramName,
      LoginName
   )
   SELECT
      @EventData.value('(/EVENT_INSTANCE/EventType)[1]',   'NVARCHAR(100)'),
      @EventData.value('(/EVENT_INSTANCE/TSQLCommand)[1]', 'NVARCHAR(MAX)'),
      @EventData,
      DB_NAME(),
      @EventData.value('(/EVENT_INSTANCE/SchemaName)[1]',  'NVARCHAR(255)'),
      @EventData.value('(/EVENT_INSTANCE/ObjectName)[1]',  'NVARCHAR(255)'),
      HOST_NAME(),
      @ip,
      PROGRAM_NAME(),
      SUSER_SNAME();
END
GO
```

Создать в VlasovaAdmin

```
create PROCEDURE TestAudit @name varchar(255),@pass varchar(255) output
as
BEGIN

          select @pass = pass
          from [dbo].[users]
          where name = @name
END
GO
```

Что-то поменять

```
SELECT *
   FROM VlasovaAuditDB.dbo.DDLEvents
   WHERE EventType = 'ALTER_PROCEDURE';
```

Теперь, чтобы сделать еще один шаг, вы можете изучить различия между исходным объектом и его самым последним состоянием, используя такой запрос:

```
WITH [Events] AS
(
   SELECT
      EventDate,
      DatabaseName,
      SchemaName,
      ObjectName,
      EventDDL,
      rnLatest = ROW_NUMBER() OVER
      (
         PARTITION BY DatabaseName, SchemaName, ObjectName
         ORDER BY     EventDate DESC
      ),
      rnEarliest = ROW_NUMBER() OVER
      (
         PARTITION BY DatabaseName, SchemaName, ObjectName
         ORDER BY     EventDate
      )
   FROM
      VlasovaAuditDB.dbo.DDLEvents
)
SELECT
   Original.DatabaseName,
   Original.SchemaName,
   Original.ObjectName,
   OriginalCode = Original.EventDDL,
   NewestCode   = COALESCE(Newest.EventDDL, ''),
   LastModified = COALESCE(Newest.EventDate, Original.EventDate)
```

```
FROM
   [Events] AS Original
LEFT OUTER JOIN
   [Events] AS Newest
   ON  Original.DatabaseName = Newest.DatabaseName
   AND Original.SchemaName   = Newest.SchemaName
   AND Original.ObjectName   = Newest.ObjectName
   AND Newest.rnEarliest = Original.rnLatest
   AND Newest.rnLatest = Original.rnEarliest
   AND Newest.rnEarliest > 1
WHERE
   Original.rnEarliest = 1;
```

Вы можете изменить DDL-триггер выше следующим образом, чтобы захватить событие ALTER_SCHEMA:

```
ALTER TRIGGER DDLTrigger_Sample
    ON DATABASE
    FOR CREATE_PROCEDURE, ALTER_PROCEDURE, DROP_PROCEDURE,
        ALTER_SCHEMA
AS
BEGIN
    -- ...
```

в SQL Server 2008 и более поздних версиях переименование можно зафиксировать с помощью события RENAME с соответствующим названием:

```
ALTER TRIGGER DDLTrigger_Sample
    ON DATABASE
    FOR CREATE_PROCEDURE, ALTER_PROCEDURE, DROP_PROCEDURE,
        ALTER_SCHEMA, RENAME
AS
BEGIN
    -- ...
```

Некоторые другие соображения:

- Возможно, вы захотите добавить процедуру очистки, которая избавится от «шума» старше <n> дней (но при этом сохранит набор важных для вас объектов).
- Чтобы убедиться, что ваш процесс аудита фиксирует все изменения, вы можете проверить дату модификации в sys.procedures. Конечно, это работает только для процедур, которые не были удалены — только если они были созданы, изменены, переименованы или перенесены в другую схему.
- Безопасность может быть проблемой, в зависимости от того, чего вы хотите достичь. Позвольте мне уточнить:

  Триггеры DDL не будут прозрачны для пользователей — во-первых, они смогут увидеть их в дереве Object Explorer, так что не будет большим секретом, что они есть и работают. Они также появляются в планах выполнения; если у пользователей включен этот параметр при создании или изменении объектов в Management Studio, они увидят план запроса для операторов, таких как INSERT AuditDB.dbo.DDLEvents.

```
ALTER TRIGGER DDLTrigger_Sample
    ON DATABASE
    WITH ENCRYPTION
    FOR -- ...
```

триггер DDL может дать сбой, если у пользователя нет разрешений INSERT для таблицы аудита.

триггер DDL будет выполняться в контексте вызывающего объекта. Вызывающий должен иметь соответствующие разрешения для создания/изменения объекта, чтобы добиться успеха (а также для срабатывания триггера в первую очередь), но может не иметь разрешений для вставки в таблицу аудита.

```
GRANT SELECT, INSERT ON OBJECT::dbo.DDLEvents TO [public];

GO
```

Могу ли я запретить системному администратору включать SQL Server xp_cmdshell?

Учитывая, что мы говорим о команде T-SQL, можем ли мы написать триггер для обработки запуска sp_configure? Соответствующим DDL-событием является **ALTER_INSTANCE** .

```
CREATE TRIGGER Stop_XP_CommandShell
ON ALL SERVER
FOR ALTER_INSTANCE
AS
BEGIN
 DECLARE @SQL NVARCHAR(4000);

 SET @SQL = (SELECT EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]',
                    'nvarchar(4000)'));

 IF (CHARINDEX('sp_configure', @SQL) > 0) AND (CHARINDEX('xp_cmdshell', @SQL) > 0)
 BEGIN
  RAISERROR('Attempt to enable xp_cmdshell detected. This operation is denied!', 16, 1);
  ROLLBACK;
 END;
END;
Проверка
EXEC sp_configure 'show advanced options', 1
GO
RECONFIGURE
GO
EXEC sp_configure 'xp_cmdshell', 1
GO
RECONFIGURE
GO
```

Несколько слов о триггерах входа в систему

https://www.mssqltips.com/sqlservertip/6103/sql-server-logon-trigger-examples/

Интуитивно думать, что триггеры входа в систему срабатывают в ответ на события входа в систему, но не очевидно, что триггеры входа срабатывают только при успешных попытках входа в систему. Другими словами, не думайте об использовании триггеров входа в систему в качестве метода аудита неудачных попыток входа в систему. Обычно триггеры входа используются для установления политик использования, таких как ограничение количества сеансов для определенных входов в систему или для аудита

успешных входов в систему.

Триггеры входа срабатывают после завершения этапа проверки подлинности при входе, но перед тем, как пользовательский сеанс реально устанавливается. Следовательно, все сообщения, которые возникают внутри триггера и обычно достигают пользователя, такие как сообщения об ошибках и сообщения от инструкции PRINT, перенаправляются в журнал ошибок SQL Server . Если проверка подлинности завершается сбоем, триггеры входа не срабатывают.

Например, в следующем коде триггер входа запрещает попытки входа на SQL Server , инициированные под именем входа *login_test* , если уже созданы три пользовательских сеанса под этим именем входа.

```
USE master;
GO
CREATE LOGIN login_test WITH PASSWORD = N'1234567' MUST_CHANGE,
   CHECK_EXPIRATION = ON;
GO
GRANT VIEW SERVER STATE TO login_test;
GO
create or alter trigger Logon_In_TimeSlot
on all server
for logon
as
begin
if(ORIGINAL_LOGIN()=N'login_test')
and(datepart(hour,getdate()) between 17 and 18)
begin
print 'Time out'
rollback
end
end

sp_readerrorlog

CREATE TABLE Event_Instance_Audit_Login
(
   [RowID] INT IDENTITY(1, 1) PRIMARY KEY,
   [EventType] VARCHAR(128) NULL,
   [PostTime] VARCHAR(128) NULL,
   [SPID] INT NULL,
   [LoginName] VARCHAR(256) NULL,
   [DatabaseName] VARCHAR(256) NULL,
   [LoginSid] VARBINARY(85) NULL

);

CREATE OR ALTER TRIGGER [TR_Login]
ON ALL SERVER
FOR LOGON
AS
DECLARE @EventData XML;
SET @EventData = EVENTDATA();

INSERT INTO VlasovaAuditDB.dbo.Event_Instance_Audit_Login
(
   [EventType] ,
   [PostTime] ,
   [SPID] ,
   [LoginName] ,
   [DatabaseName] ,
   [LoginSid]
)
VALUES
(@EventData.value('(/EVENT_INSTANCE/EventType)[1]', 'VARCHAR(128)'),
```

```
 @EventData.value('(/EVENT_INSTANCE/PostTime)[1]', 'VARCHAR(128)'),
 @EventData.value('(/EVENT_INSTANCE/SPID)[1]', 'INT'),
 @EventData.value('(/EVENT_INSTANCE/LoginName)[1]', 'VARCHAR(256)'),
 @EventData.value('(/EVENT_INSTANCE/DatabaseName)[1]', 'VARCHAR(256)'),
 @EventData.value('(/EVENT_INSTANCE/LoginSid)[1]', 'VARBINARY(85)'));
GO
```

Права на запись в таблицу

```
SELECT * FROM sys.dm_exec_sessions
        WHERE is_user_process = 1 AND
           original_login_name = N'login_test'
kill 52
drop trigger [TR_Login ] ON ALL SERVER


USE master;
GO
CREATE LOGIN login_test WITH PASSWORD = N'1234567' MUST_CHANGE,
   CHECK_EXPIRATION = ON;
GO
GRANT VIEW SERVER STATE TO login_test;
GO
alter TRIGGER connection_limit_trigger
ON ALL SERVER WITH EXECUTE AS N'login_test'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN()= N'login_test' AND
   (SELECT COUNT(*) FROM sys.dm_exec_sessions
        WHERE is_user_process = 1 AND
           original_login_name = N'login_test') > 3
   print 'Error';
END;
```