Журналирование операций с таблицами в БД MS SQL Server

Сопроводительная документация к скриптам

- pmate.log\createlog.py
- pmate.log\logtable.sql

Задача сформулирована так:
Есть некая система “Project Mate”, реализованная как клиент-сервер приложение. Толстый клиент работает с данными в БД MS SQL Server.
Нужно получить возможность отправлять сообщения (абстрактные) в стиле документооборота, источником которых являются изменения в БД системы.
В качестве генератора сообщений выбран механизм Workflow MS Sharepoint.

Workflow типа “диспетчер” будет регулярно опрашивать таблицу “журнал” в БД, получая из нее идентификаторы новых/обновленных/удаленных записей, вида
“имя таблицы;имя ключевого поля;значение ключа;тип операции;время операции”.
После чего сможет выбрать из таблиц БД прикладные данные, следуя схеме БД. Имея эти данные, workflow может сформировать необходимые сообщения.

Решение задачи журналирования

В БД создается специальная таблица “журнал”, для сбора сведений о изменениях в прикладных таблицах. Сведения собираются в журнал с помощью триггеров на изменения в отслеживаемых таблицах.

Создание журнала

Для создания таблицы журнала надо выполнить в целевой БД код SQL из прилагаемого файла **logtable.sql**
По умолчанию, в БД **PJM6** будет создано два объекта:

- сиквенс [nintex].[pmateLogSequence]
- таблица [nintex].[pmateLog]

Если журнал надо создать в другой БД, отредактируйте строку 11 скрипта, заменив текст
`USE [PJM6]`
на требуемый вариант.

Поскольку в первой части скрипта происходит удаление журнала, при первом запуске скрипта система будет ругаться на невозможность найти удаляемые объекты - это нормально. Такие сообщения о ошибках можно игнорировать.

После создания таблицы журнала надо создать набор триггеров, привязанных к отслеживаемым таблицам. Для этого надо выполнить код двух скриптов.
Первый скрипт содержится в файле **createlog.py**
и результатом его выполнения будет второй скрипт - код SQL для запуска внутри целевой БД.
Перед выполнением createlog.py проверьте, чтобы **входные параметры** соответствовали требованиям. Входные параметры записаны в строках 24-37 скрипта.
Пример:

```
DBNAME = 'PJM6'
LOGTABLE = '[nintex].[pmateLog]'
TRIGGERPREFIX = 'nintex_logTrigger_'
TABLESINFO = '''
      dbo.UserRole;UserID,RoleID
      dbo.Statement;ID
      dbo.Invoice;ID
      dbo.Payment;ID
      dbo.TimeEntry;ID
      dbo.Expense;ID
      dbo.Project;ID
      dbo.Calculation;ID'''
```

Здесь DBNAME - это имя БД в которой будут создаваться триггеры.
LOGTABLE - это имя таблицы журнала, в которую будут записываться события.
TRIGGERPREFIX - это первая, общая часть имени для создаваемых триггеров. Вторая часть имени представляет собой имя таблицы для которой создается триггер.
TABLESINFO - это список таблиц, для которых надо отслеживать изменения в журнале. Каждая строка содержит сведения об одной таблице в формате
имяСхемы.ИмяТаблицы;имяПоляПервичногоКлюча
Если первичный ключ составной, то имяПоляПервичногоКлюча записывается в виде
имяПервогоПоля,имяВторогоПоля,имяТретьегоПоля
причем, больше трех полей в составном ключе быть не может - не предусмотрено.

После проверки входных параметров, выполните скрипт **createlog.py**
Для этого можно воспользоваться онлайн сервисами:

- http://repl.it/languages/Python
- http://www.compileonline.com/execute_python_online.php
- http://ideone.com/
- http://codepad.org/

или установить интерпретатор Python на локальную машину

- http://www.python.org/download/releases/2.7.6/

Результатом выполнения createlog.py будет код SQL вида:

```
USE [PJM6]
GO
...
DROP TRIGGER [dbo].[nintex_logTrigger_UserRole]
GO
...
...
CREATE TRIGGER [dbo].[nintex_logTrigger_UserRole]
…
GO
```

Этот код надо выполнить в целевой БД. В базе будут созданы триггеры, по одному на каждую отслеживаемую таблицу. Как и в случае с созданием таблицы журнала, при первом запуске система может ругаться на невозможность удалить несуществующие объекты - это нормально и можно игнорировать.

Обязательно сохраните первую часть скрипта, содержащую инструкции DROP! При возникновении любых проблем с работоспособностью прикладной системы, первым делом удалите триггеры журналирования - с помощью этой сохраненной части скрипта.

Структура таблицы журнала

```
CREATE TABLE [nintex].[pmateLog](
    [id] [int] PRIMARY KEY CLUSTERED
        DEFAULT (NEXT VALUE FOR [nintex].[pmateLogSequence]),
    [tableName] [nvarchar](80) NOT NULL,
    [keyName] [nvarchar](80) NOT NULL,
    [keyName2] [nvarchar](80) NULL,
    [keyName3] [nvarchar](80) NULL,
    [keyValue] [nvarchar](80) NOT NULL,
    [keyValue2] [nvarchar](80) NULL,
    [keyValue3] [nvarchar](80) NULL,
    [opName] [varchar](7) NOT NULL,
    [opDate] [datetime] NOT NULL DEFAULT (getdate()),
    [recStatus] [nvarchar](50) NULL )
```

Для формирования ключей таблицы используется Sequence вместо IDENTITY, по причине того, что использование identity может влиять на работу прикладных программ системы.

```
CREATE SEQUENCE [nintex].[pmateLogSequence]
    AS [int]
    START WITH -2147483648
    INCREMENT BY 1
    MINVALUE -2147483648
```

```
    MAXVALUE 2147483647
    CACHE
```

Описание полей таблицы журнала

[id] - Суррогатный ключ для ссылок на конкретную запись журнала (операцию с данными системы). При вставке записи в журнал, ключевое значение записывается автоматически с помощью констрейнта “`DEFAULT (NEXT VALUE FOR [nintex].[pmateLogSequence])`” берущего значения из сиквенса pmateLogSequence. При ежедневном создании 1000 000 записей журнала, емкости сиквенса хватит на 10 лет, после чего журнал необходимо пересоздать.

[tableName] - Название таблицы, в которой были обновлены данные. Пример типичного значения “[dbo].[Accounts]”. Скобочки добавлены для удобства последующего использования имени таблицы в запросах SQL.

[keyName] - Название ключевого поля в таблице, в которой были обновлены данные. Если у таблицы ключ составной, то понадобятся имена полей из keyName2 и, возможно, keyName3. Пример типичного значения “[ID]”.

[keyName2] - Название второго поля, входящего в составной ключ, если ключ является составным.

[keyName3] - Название третьего поля, входящего в составной ключ, если ключ является составным.

[keyValue] - Значение ключевого поля для обновленной записи в отслеживаемой таблице. Пример типичного значения “123”. В сочетании с именем таблицы из tableName и именем ключевого поля из keyName, keyValue позволяет однозначно определить измененную запись в отслеживаемой таблице.

[keyValue2] - Значение второго поля, входящего в составной ключ.

[keyValue3] - Значение третьего поля, входящего в составной ключ.

[opName] - Название операции, произведенной над отслеживаемой таблицей. Одно из трех: “insert”, “update”, “delete”.

[opDate] - Временной штамп, показывающий время занесения записи в журнал. Предназначен для использования в периодической очистке журнала.

[recStatus] - Произвольная строка, используемая системой чтения журнала. Начальное значение - NULL. Чтобы не обрабатывать повторно уже отработанные записи журнала, система чтения может записывать в это поле условное значение, скажем “1”. Тогда, применив простой фильтр, можно отсечь из выборки уже отработанные записи.

Примеры запросов к журналу

Выборка всех записей из журнала с добавлением вычисляемого столбца “minutesago”, показывающего - сколько минут назад была совершена каждая операция:

```
select DATEDIFF(MINUTE, opDate, getdate()) as minutesago, * from
nintex.pmateLog
```

Удаление из журнала записей сделанных более 60 минут назад:

```
delete from [nintex].[pmateLog] where DATEDIFF(MINUTE, [opDate], GETDATE()) >
60
```

Обновление статуса записей в журнале, установка статуса “1” для записей старше 30 минут:

```
update nintex.pmateLog set recStatus = '1' where DATEDIFF(MINUTE, [opDate],
GETDATE()) > 30
```

Выборка всех записей журнала, для которых статус не равен “1”:

```
select * from nintex.pmateLog where recStatus not like '1' or recStatus is
null
```

Выборка записей журнала на основе конкретной даты и времени. Будут выбраны все записи, сделанные до момента времени 2014-03-08 16:55:59:

```
select * from nintex.pmateLog where opDate < '2014-03-08 16:55:59'
```

Заполнение журнала, триггеры

Как уже было сказано, таблицу журнала заполняют триггеры, создаваемые по одной штуке на каждую отслеживаемую таблицу. Текст триггеров генерируется скриптом createlog.py используя входные параметры, записываемые в заголовке этого скрипта. Все триггеры сделаны по одному шаблону, за основу которого был взят триггер из файла trigger.sql

Код типового триггера выглядит так:

```
CREATE TRIGGER [dbo].[nintex_logTrigger_Calculation]
   ON  [dbo].[Calculation]
   AFTER INSERT,DELETE,UPDATE
AS
```

```
BEGIN
    SET NOCOUNT ON;
    DECLARE @i INT, @d INT;
    SELECT @i = COUNT(*) FROM inserted;
    SELECT @d = COUNT(*) FROM deleted;
    IF @i + @d > 0
    BEGIN
        IF @i > 0 AND @d = 0
            -- insert
            INSERT INTO [nintex].[pmateLog](tableName, keyName,  keyValue,
opName)
            SELECT '[dbo].[Calculation]' as tn, '[ID]' as kn,  [ID] as kv,
'insert' as opn
            from inserted
        IF @i > 0 AND @d > 0
            -- update
            INSERT INTO [nintex].[pmateLog](tableName, keyName,  keyValue,
opName)
            SELECT '[dbo].[Calculation]' as tn, '[ID]' as kn,  [ID] as kv,
'update' as opn
            from inserted
        IF @i = 0 AND @d > 0
            -- delete
            INSERT INTO [nintex].[pmateLog](tableName, keyName,  keyValue,
opName)
            SELECT '[dbo].[Calculation]' as tn, '[ID]' as kn,  [ID] as kv,
'delete' as opn
            from deleted
    END
END
GO
```

Имя триггера [dbo].[nintex_logTrigger_Calculation] составляется из имени отслеживаемой таблицы [dbo].[Calculation] и входного параметра TRIGGERPREFIX = 'nintex_logTrigger_' Вся суть триггера заключается в том, что он извлекает значение первичного ключа измененной таблицы из псевдотаблиц inserted или deleted, после чего записывает его в журнал [nintex].[pmateLog] наряду с именами отслеживаемой таблицы и ее первичного ключа.

Возможное влияние на прикладную систему

Наличие в БД таблицы журнала и сиквенса влияния на прикладные системы не оказывает никакого. Разве что, для хранения записей журнала нужно место на диске. Если журнал периодически очищать от старых записей, места нужно совсем немного.

Иначе обстоит дело с триггерами. В архитектуре MS SQL Server заложено решение, которое можно озвучить так: если в триггере произошла ошибка, вся транзакция откатывается. Другими словами, невозможно внутри триггера обработать ошибку так, чтобы прикладная транзакция не отвалилась.
Из этого следует очень простой и печальный вывод - любой сбой в системе журналирования приводит к невозможности внесения изменений в отслеживаемые таблицы.
Какие могут быть сбои в системе журналирования?
Поскольку система очень простая, то такие сбои могут быть только теоретически, но все же: что-то случилось с сиквенсом - он был переинициализирован или в нем кончились значения (4 миллиарда). В этом случае в таблицу журнала невозможно добавить запись, что ведет к сбоям во всех триггерах журналирования.
Или, кто-то удалил, изменил, заблокировал таблицу журнала - результат тот же, все триггеры журналирования отваливаются по ошибке записи в журнал.

Более вероятные ошибки связаны с человеческим фактором. Самое вероятное - неправильно заданы параметры в скрипте createlog.py, что ведет к некорректному тексту создаваемых триггеров. В этом случае сбоить будут только неправильные триггеры, правильные продолжат работать нормально.

Что делать в результате обнаружения сбоев? Удалить все триггеры журналирования. Для этого у вас под рукой должен быть файл с текстом SQL, в котором зафиксированы все созданные ранее триггеры, о чем подробно написано выше, в части “Создание журнала”.

доп.инфо

https://www.google.ru/search?q=асинхронный+триггер+sql

MSSQL 2005 (Yukon) – работа с очередями и асинхронная обработка данных
http://www.rsdn.ru/article/db/Yukon_Async.xml

http://blogs.msdn.com/b/chaks/archive/2011/05/02/concept-leverage-sharepoint-workflows-with-external-lists-part-1.aspx

http://www.t-sql.ru/post/sp_settriggerorder.aspx

http://www.sql.ru/faq/faq_topic.aspx?fid=578 (Суть проблемы в том, что внешние по отношении к триггеру программы пытаются получить @@IDENTITY последней добавленной записи, а при осуществлении добавления записи любой таблицы внутри триггера это значение заменяется новым.)

You can use the inserted and deleted tables to see what changes were made to the table.
For an UPDATE, the deleted table contains the old version of the row, and inserted the new version.
DELETE and INSERT use their own table as you would expect

http://habrahabr.ru/post/111207/
Change tracking; change data capture

Реализация инкрементной загрузки с использованием Change Data Capture в SQL Server
http://www.sql.ru/blogs/kab/1591

Механизмы аудита и отслеживания изменений данных в SQL Server 2008
http://blogs.technet.com/b/isv_team/archive/2008/06/05/sql-server-2008-audit-and-change-capture.aspx

Система отслеживания измененных данных SQL Server 2008 R2
http://technet.microsoft.com/ru-ru/library/bb522489%28v=sql.105%29.aspx

https://www.google.ru/search?q=отслеживание+изменений+sql+server

MSDN Blogs > SQLClub > CDC http://blogs.msdn.com/b/alexejs/archive/2009/08/07/cdc.aspx

Отслеживание изменений в SQL Server 2008 http://habrahabr.ru/post/111207/