Technical Papers

Planning Suite

Leveraging SQL triggers

May 2015

Table of Contents

Overview

This document provides examples* of leveraging database triggers in the development of applications with the Planning Suite.

The examples fall into the following three categories:

- Fields conditional update — To manage logically-linked fields in a record (E.g. quantity, price, amount).
- Data manipulation — To allow users performing operations on large volumes of data (E.g. budget initialization, copy of a version, etc.).
- Audit trails — To track data changes with full details and history.

Important! These examples just want to show the way, they don't have the ambition of being 'perfect' in terms of SQL coding.

* based on MS SQL Server

Triggers

Database trigger – a definition

A trigger is a special kind of a store procedure that executes in response to certain action on the table like insertion, deletion or update of data. It is a database object which is bound to a table and is executed automatically. You can't explicitly invoke triggers. The only way to do this is by performing the required action on the table that they are assigned to. In Appendix A you can find a list of links to useful materials on this subject. Multiple triggers can be assigned to the same table.

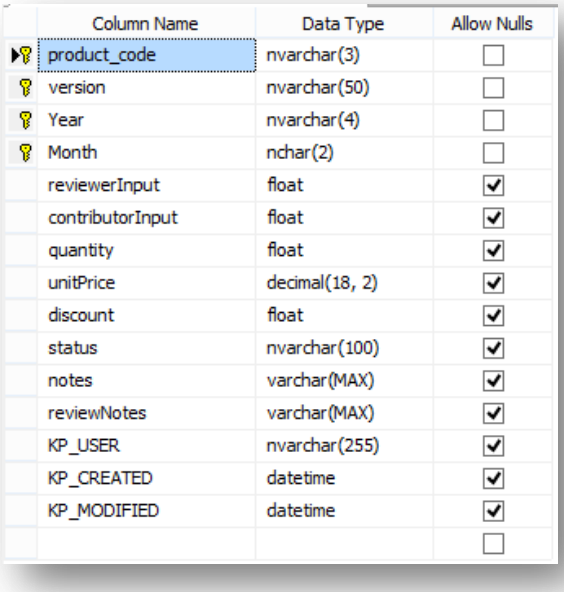
Creating a trigger in MS SQL Server

You can find clear instructions on how to create a Transact-SQL DML trigger by using SQL Server Management Studio and by using the Transact-SQL CREATE TRIGGER statement here:

https://msdn.microsoft.com/en-US/library/ms190227.aspx

By default the page refers to SQL 2014 but you can change the version with the drop-down under the title.

Leveraging triggers - fields conditional update



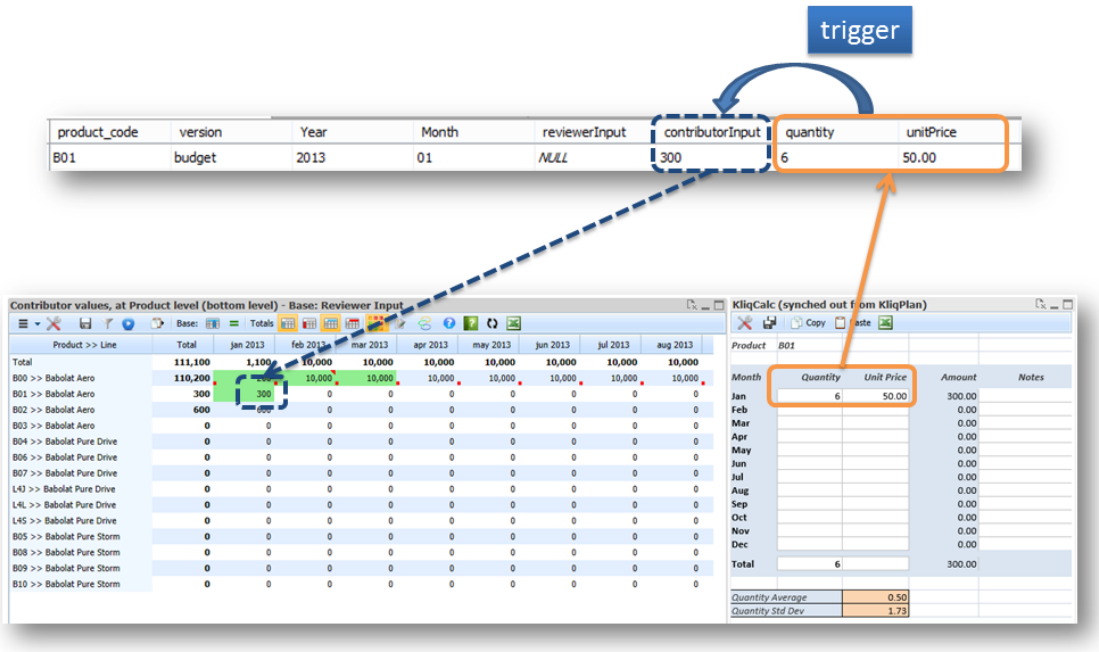
Column Name	Data Type	Allow Nulls
product_code	nvarchar(3)	☐
version	nvarchar(50)	☐
Year	nvarchar(4)	☐
Month	nchar(2)	☐
reviewerInput	float	☑
contributorInput	float	☑
quantity	float	☑
unitPrice	decimal(18, 2)	☑
discount	float	☑
status	nvarchar(100)	☑
notes	varchar(MAX)	☑
reviewNotes	varchar(MAX)	☑
KP_USER	nvarchar(255)	☑
KP_CREATED	datetime	☑
KP_MODIFIED	datetime	☑
		☐

We can use a trigger to update fields which are *logically* linked. In the case of the table *budget_month_prod* in the DemoTennis database, we may want to:

- every time the fields *quantity* or *unitPrice* are updated, recalculate *contributorInput* as *quantity* * *unitPrice*
- Every time the field *contributorInput* is updated, recalculate *quantity* as *contributorInput* / *unitPrice*

In the trigger we can use the function UPDATE() to determine which field was updated. Using the virtual table *inserted* created by SQL Server and containing all the updated records, we then do the recalculation using a *merge* instruction.

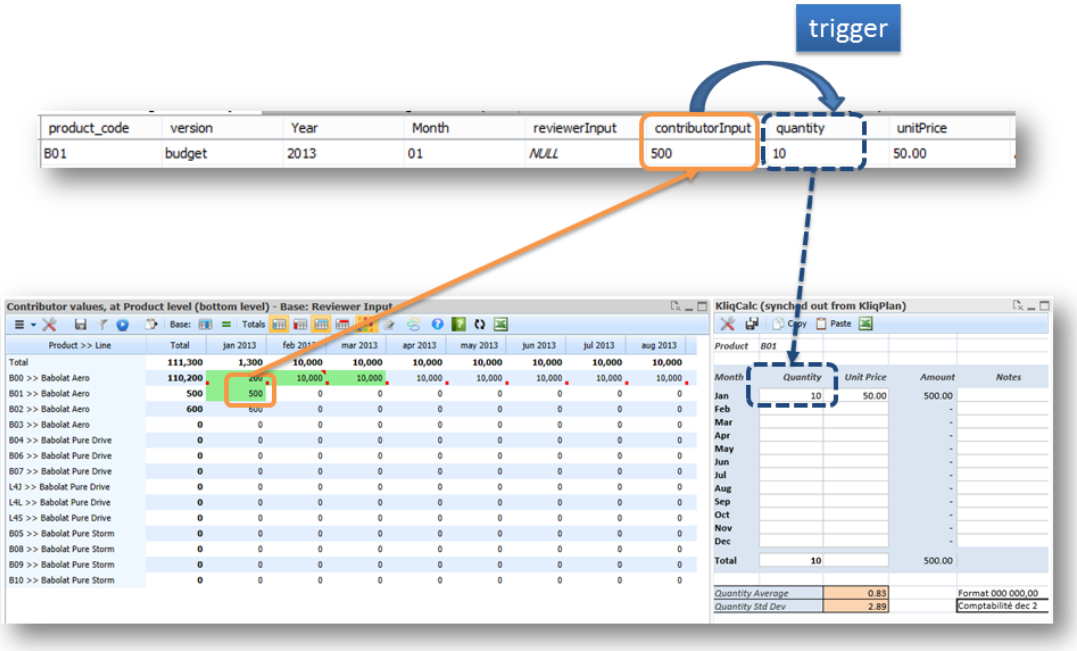
Updating *contributorInput*




```
IF ( UPDATE (quantity)  or UPDATE (unitPrice))
BEGIN
       merge into budget month prod as target
       using (SELECT inserted.product code,inserted.Month,inserted.Year,inserted.version,
       inserted.contributorInput, inserted.quantity, inserted.unitPrice from inserted)as source
          on target.product_code = source.product_code
               AND target.Month = source.Month
               AND target.Year = source.Year
               AND target.version = source.version
       when matched
               then update set
               contributorInput= case when source.unitPrice is not NULL and source.quantity is not
       NULL then source.quantity * source.unitPrice else source.contributorInput end;

END;
```

Updating *quantity*




```
IF ( UPDATE (contributorInput) )
BEGIN
        merge into budget month prod as target
        using (SELECT inserted.product_code,inserted.Month,inserted.Year,inserted.version,
        inserted.contributorInput, inserted.quantity, inserted.unitPrice from inserted)
        as source
           on target.product code = source.product code
                AND target.Month = source.Month
                AND target.Year = source.Year
                AND target.version = source.version
        when matched
                then update set
                quantity = case when source.unitPrice <> 0 then source.contributorInput /
        source.unitPrice else source.quantity end;

END;
```

You can find the trigger complete source in Appendix B – fields conditional update trigger.

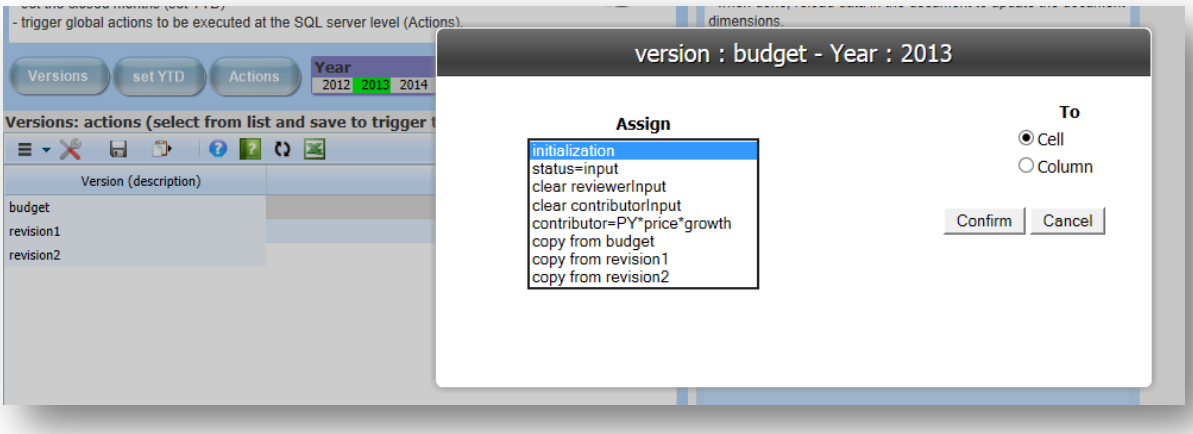
Leveraging triggers – data manipulation

Although it is possible to manipulate data interactively in KliqPlan/KliqCalc, whenever the data volume is significant it is a best practice to leverage the power of the backend database by giving the user the ability of launching manipulations of selected data. Examples of such manipulations are initializing budget data or make a full copy of a version.

This is can be accomplished by using KliqPlan to update a text field (e.g. *action*) in an *actions* table which has a trigger that will perform different actions based on the field value.

	Column Name	Data Type	Allow Nulls
🔑	Year	nchar(4)	☐
🔑	version	nvarchar(50)	☐
	action	nvarchar(100)	☑
	KP_USER	nvarchar(255)	☑
	KP_CREATED	datetime	☑
	KP_MODIFIED	datetime	☑

In KliqPlan you can set the list of possible *actions* by setting the input *Type* to *List* and setting the list of possible values in the properties *List Values* and *Cell List Values*.



The list of actions could be *static*:

```
='initialization;status=input;clear reviewerInput;clear contributorInput;contributor=PY*price*growth'
& concat(';copy from ' & version )
```

Or *dynamic* (for example to give the possibility of choosing from which version we want to copy):

```
='initialization;status=input;clear reviewerInput;clear contributorInput;contributor=PY*price*growth'
& concat(';copy from ' & version )
```

The primary keys of the *actions* table will allow the trigger to perform the selected action only inside a given perimeter (version=budget, Year=2015).

You can find the trigger complete source in Appendix C – data manipulation trigger

Let's examine in detail some actions.

Action: status=input

In this case we are setting the value *input* in the field *status* of the table *budget_month_prod*.

- As usual in SQL Server we use the virtual table *inserted* which contains all the inserted/modified records.
- We join *inserted* with *budget_month_prod* on *Year* and *version*
- We use the *GETDATE()* SQL function to set the field *KP_MODIFIED*

```
-- action:     status=input
UPDATE p
  SET status='input', KP_USER=i.KP_USER ,KP_MODIFIED=GETDATE()
  FROM budget month prod AS p
  INNER JOIN inserted AS i
  ON p.Year = i.Year and p.version = i.version
  where i.action = 'status=input';
```

Action: contributor=PY*price*growth

In this case the code is a bit more complex because instead of using a single *UPDATE* statement we fetch one by one the modified records from the actions table and for each one we use the view *w_budget_init* (please see the *DemoTennis* database for details) where we make the calculation by combining the previous year value from the *sales* table with the *priceChange* from *budget_quarter_line_priceChange* and *growth* from *budget_year_brand_growth*.

```
    DECLARE cur CURSOR FORWARD ONLY READ ONLY LOCAL FOR
        SELECT Year, version, KP USER
        FROM INSERTED where action = 'contributor=PY*price*growth';

    OPEN cur

    FETCH NEXT FROM cur INTO @curYEAR, @curVERSION, @curUSER

    WHILE @@FETCH_STATUS = 0 BEGIN
…
```

Action: version copy

Having generated the list of possible actions with the QlikView expression:

```
='initialization;status=input;clear reviewerInput;clear contributorInput;contributor=PY*price*growth'
& concat(';copy from ' & version )
```

We then get, in the trigger code, the name of the version to copy from by extracting from the *action* value the text starting from the 11th position (after *copy from*) using the *SUBSTRING* function:

```
DECLARE cur CURSOR FORWARD ONLY READ ONLY LOCAL FOR
      SELECT Year, version, SUBSTRING(action,11,50), KP USER
      FROM INSERTED where SUBSTRING(action,1,10) = 'copy from ';

  OPEN cur

  FETCH NEXT FROM cur INTO @copyFromYEAR, @copyToVERSION, @copyfromVERSION, @curUSER3
```

Leveraging triggers - audit trail

In accounting, an audit trail is the sequence of paperwork that validates or invalidates accounting entries. In computing, the term is also used for an electronic or paper log used to track computer activity.

The *Planning Suite* uses the standard audit trail fields KP_USER, KP_CREATED and KP_MODIFIED to register in each record the name of the user who made the last modification and a timestamp for the record creation and last change. There is no *history* of changes.

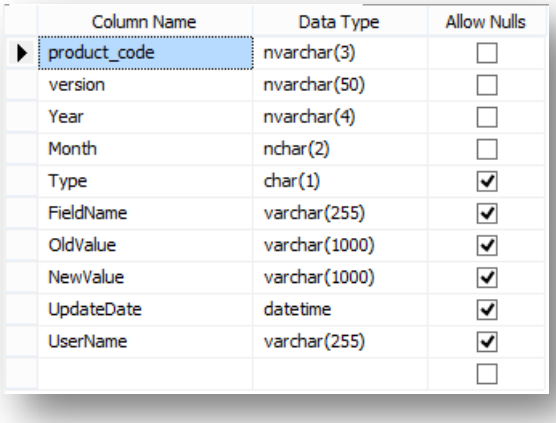
If you want to manage a more detailed audit trail you can either use specialized software tools (E.g. the one from ApexSQL - www.apexsql.com) or implement a trigger on the table to store detailed audit trail information in a different table.

As an example this chapter illustrates a parametric trigger which parses automatically the SQL meta-data to selectively monitor changes field by field. The code of the trigger (fully described in appendix D) is the same for every table; you simply have to set in it the parameter *@TableName* with the source table name and *@FieldList* with the fields (excluded the primay keys) you want to monitor. The audit trail table where the changes are recorded must have a standard structure illustrated in the example hereinafter.

Example: creating an audit trail trigger on table *budget_month_prod* in the DemoTennis database.

- Create the audit trail table called *AUDIT_budget_month_prod*. The table must have a set of fields corresponding to the *budget_month_prod* table primary keys (product_code, version, Year and Month) plus:

Field name	Field type	Description
Type	char(1)	Type of action: ▪ I for inserted ▪ U for updated ▪ D for deleted
FieldName	varchar(255)	Name of the field
OldValue	varchar(1000)	Previous field value
NewValue	varchar(1000)	New field value
UpdateDate	datetime	Change timestamp
UserName	varchar(255)	User names



Column Name	Data Type	Allow Nulls
product_code	nvarchar(3)	☐
version	nvarchar(50)	☐
Year	nvarchar(4)	☐
Month	nchar(2)	☐
Type	char(1)	☑
FieldName	varchar(255)	☑
OldValue	varchar(1000)	☑
NewValue	varchar(1000)	☑
UpdateDate	datetime	☑
UserName	varchar(255)	☑
		☐

- Create on the *budget_month_prod* a new trigger:
  a. Paste the code from Appendix D – audit trail trigger
  b. Set the variable *@TableName* to *budget_month_prod*:

```
-- table name
select @TableName = 'budget_month_prod'
```

  c. Set the variable *@FieldList* to the list of fields you want to monitor (do not include primary keys):

```
-- list of fields to be monitored
INSERT INTO @FieldList VALUES('reviewerInput'),('contributorInput');
```

This is an example of what the *AUDIT_budget_month_prod* table will contain:

product_code	version	Year	Month	Type	FieldName	OldValue	NewValue	UpdateDate	UserName
300	budget	2013	01	U	reviewerInput	500	100	2015-05-25 13:43:26.573	server\Edward
B01	budget	2013	03	U	reviewerInput	-10.32	111	2015-05-25 13:56:05.510	server\Edward
B01	budget	2013	03	U	contributorInput	*NULL*	23	2015-05-25 13:56:22.880	server\Susan

The trigger described in Appendix C, if the table *AUDIT_[table name]* does not exist, will automatically use (if existing) a 'catch-all' table called *AUDIT_DATABASE* with the following structure:

Column Name	Data Type	Allow Nulls
Type	char(1)	☑
TableName	varchar(128)	☑
PK	varchar(1000)	☑
FieldName	varchar(128)	☑
OldValue	varchar(1000)	☑
NewValue	varchar(1000)	☑
UpdateDate	datetime	☑
UserName	varchar(128)	☑

The field *PK* will contain the primary keys values.

Type	TableName	PK	FieldName	OldValue	NewValue	UpdateDate	UserName
U	budget_month_prod	Month=01;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=02;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=03;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=04;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=05;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=06;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=07;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=08;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=09;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=10;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=11;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan
U	budget_month_prod	Month=12;product_code=H08;version=budget ;Year=2013;	contributorInput	*NULL*	1000	2015-05-27 08:...	Susan

You can find the trigger complete source in Appendix D – audit trail trigger.

How to deactivate the audit trail trigger

Since the trigger checks automatically if the tables AUDIT_[table name] and AUDIT_DATABASE exist, in order to temporary disable it, it is sufficient to rename the two tables.

Appendix A – Resources

Why use triggers in Microsoft SQL Server?	http://searchsqlserver.techtarget.com/feature/Why-use-triggers-in-Microsoft-SQL-Server
MSDN CREATE TRIGGER (Transact-SQL)	https://msdn.microsoft.com/en-us/library/ms189799.aspx
Exploring SQL Server Triggers	https://msdn.microsoft.com/en-us/magazine/cc164047.aspx
Overview of SQL Server database Triggers	http://www.codeproject.com/Articles/38808/Overview-of-SQL-Server-database-Triggers
Generic audit trail trigger by Nigel Rivett	http://www.nigelrivett.net/AuditTrailTrigger.html
Using Triggers in Oracle	https://docs.oracle.com/database/121/TDDDG/tdddg_triggers.htm#TDDDG50000

Appendix B – fields conditional update trigger

```
USE [DemoTennis]
GO

/****** Object:  Trigger [dbo].[budget_month_prod_UPDATE]    ******/
SET ANSI NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO


CREATE trigger [dbo].[budget month prod UPDATE] on [dbo].[budget month prod] for update
as

IF ( UPDATE (contributorInput) )
BEGIN
       merge into budget month prod as target
       using (SELECT inserted.product code,inserted.Month,inserted.Year,inserted.version,
       inserted.contributorInput, inserted.quantity, inserted.unitPrice from inserted)
       as source
          on target.product_code = source.product_code
               AND target.Month = source.Month
               AND target.Year = source.Year
               AND target.version = source.version
       when matched
               then update set
               quantity = case when source.unitPrice <> 0 then source.contributorInput /
       source.unitPrice else source.quantity end;

END;

IF ( UPDATE (quantity)  or UPDATE (unitPrice))
BEGIN
       merge into budget month prod as target
       using (SELECT inserted.product_code,inserted.Month,inserted.Year,inserted.version,
       inserted.contributorInput, inserted.quantity, inserted.unitPrice from inserted)as source
          on target.product_code = source.product_code
               AND target.Month = source.Month
               AND target.Year = source.Year
               AND target.version = source.version
       when matched
               then update set
               contributorInput= case when source.unitPrice is not NULL and source.quantity is not
       NULL then source.quantity * source.unitPrice else source.contributorInput end;

END;


GO
```

Appendix C – data manipulation trigger

```
USE [DemoTennis]
GO

/****** Object:  Trigger [dbo].[budget_actions_EXEC]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED IDENTIFIER ON
GO


CREATE TRIGGER [dbo].[budget actions EXEC]
   ON  [dbo].[budget actions]
   AFTER UPDATE,INSERT
AS
BEGIN
        SET NOCOUNT ON;


-- action:     status=input
UPDATE p
  SET status='input', KP_USER=i.KP_USER ,KP_MODIFIED=GETDATE()
  FROM budget month prod AS p
  INNER JOIN inserted AS i
  ON p.Year = i.Year and p.version = i.version
  where i.action = 'status=input';


-- action:     clear reviewerInput

UPDATE p
  SET reviewerInput=NULL, KP_USER=i.KP_USER , KP_MODIFIED=GETDATE()
  FROM budget_month_prod AS p
  INNER JOIN inserted AS i
  ON p.Year = i.Year and p.version = i.version
  where i.action = 'clear reviewerInput';



-- action:     clear contributorInput

UPDATE p
  SET contributorInput=NULL,KP_USER=i.KP_USER, KP_MODIFIED=GETDATE()
  FROM budget_month_prod AS p
  INNER JOIN inserted AS i
  ON p.Year = i.Year and p.version = i.version
  where i.action = 'clear contributorInput';



-- action:     contributor=PY*price*growth

SET NOCOUNT ON
    SET XACT_ABORT ON

    DECLARE @curUSER VARCHAR(100)
    DECLARE @curYEAR VARCHAR(4)
    DECLARE @curVERSION VARCHAR(100)
       DECLARE @curtime smalldatetime = GETDATE()

    DECLARE cur CURSOR FORWARD_ONLY READ_ONLY LOCAL FOR
        SELECT Year, version, KP USER
        FROM INSERTED where action = 'contributor=PY*price*growth';

    OPEN cur

    FETCH NEXT FROM cur INTO @curYEAR, @curVERSION, @curUSER
```

```
    WHILE @@FETCH_STATUS = 0 BEGIN

               merge into budget month prod as target
               using (SELECT Month,product code,Year,version,initValue from w budget init where
Year=@curYEAR and version = @curVERSION)
               as source
                  on target.Month = source.Month
                       AND target.product_code = source.product_code
                       AND target.Year = source.Year
                       AND target.version = source.version
               when not matched by target
                       then insert (Month,product_code,Year,version,contributorInput, KP_USER,
KP_CREATED) values (Month,product_code,Year,version,initValue,@curUSER,@curtime )
               when matched
                       then update set contributorInput = source.initValue, KP USER=@curUSER,
KP_MODIFIED=@curtime;

            FETCH NEXT FROM cur INTO  @curYEAR, @curVERSION, @curUSER

    END

    CLOSE cur
    DEALLOCATE cur


-- action:     initialization

SET NOCOUNT ON
    SET XACT_ABORT ON

    DECLARE @curUSER2 VARCHAR(100)
    DECLARE @curYEAR2 VARCHAR(4)
    DECLARE @curVERSION2 VARCHAR(100)
       DECLARE @curtime2 smalldatetime = GETDATE()

    DECLARE cur CURSOR FORWARD ONLY READ ONLY LOCAL FOR
        SELECT Year, version, KP USER
        FROM INSERTED where action = 'initialization';

    OPEN cur

    FETCH NEXT FROM cur INTO @curYEAR2, @curVERSION2, @curUSER2

    WHILE @@FETCH_STATUS = 0 BEGIN

               merge into budget_month_prod as target
               using (SELECT Month,product code,Year,version,initValue from w budget init where
Year=@curYEAR2 and version = @curVERSION2)
               as source
                  on target.Month = source.Month
                       AND target.product_code = source.product_code
                       AND target.Year = source.Year
                       AND target.version = source.version
               when not matched by target
                       then insert (Month,product code,Year,version,status,KP CREATED,KP USER) values
(Month,product_code,Year,version,'input' ,@curtime2,@curUSER2)
               when matched
                       then update set  KP_MODIFIED=@curtime2, KP_USER= @curUSER2, status='input',
contributorInput = NULL, reviewerInput = NULL, KP CREATED= NULL,notes= NULL,reviewNotes= NULL;

            FETCH NEXT FROM cur INTO  @curYEAR2, @curVERSION2, @curUSER2

    END

    CLOSE cur
    DEALLOCATE cur


-- action:     copy from

SET NOCOUNT ON
```

```
    SET XACT_ABORT ON

    DECLARE @curUSER3 VARCHAR(100)
    DECLARE @copyFromYEAR VARCHAR(4)
    DECLARE @copyToVERSION VARCHAR(100)
    DECLARE @copyfromVERSION VARCHAR(100)
       DECLARE @curtime3 smalldatetime = GETDATE()

    DECLARE cur CURSOR FORWARD_ONLY READ_ONLY LOCAL FOR
        SELECT Year, version, SUBSTRING(action,11,50), KP USER
        FROM INSERTED where SUBSTRING(action,1,10) = 'copy from ';

    OPEN cur

    FETCH NEXT FROM cur INTO @copyFromYEAR, @copyToVERSION, @copyfromVERSION, @curUSER3

    WHILE @@FETCH_STATUS = 0 BEGIN

               merge into budget_month_prod as target
               using (SELECT Month,product_code,Year,@copyToVERSION as
version,contributorInput,reviewerInput from budget month prod where Year=@copyFromYEAR and version =
@copyfromVERSION)
               as source
                  on target.Month = source.Month
                       AND target.product_code = source.product_code
                       AND target.Year = source.Year
                       AND target.version = source.version
               when not matched by target
                       then insert
(status,Month,product_code,Year,version,contributorInput,reviewerInput,KP_CREATED,KP_USER) values
('input',
Month,product code,Year,@copyToVERSION,source.contributorInput,source.reviewerInput,@curtime3,@curUS
ER3)
               when matched
                       then update set  KP_MODIFIED=@curtime3, KP_USER= @curUSER3, status='input',
contributorInput = source.contributorInput, reviewerInput = source.reviewerInput, KP_CREATED=
NULL,notes= NULL,reviewNotes= NULL;

           FETCH NEXT FROM cur INTO  @copyFromYEAR, @copyToVERSION, @copyfromVERSION, @curUSER3

    END

    CLOSE cur
    DEALLOCATE cur


-- clear budget_actions table
  update budget actions set action = NULL

END

GO
```

Appendix D – audit trail trigger

This code is based on the example by Nigel Rivett – see Appendix A – Resources.

```
USE [DemoTennis]
GO

/****** Object:  Trigger [dbo].[budget month prod AUDIT]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED IDENTIFIER ON
GO


CREATE trigger [dbo].[budget_month_prod_AUDIT] on [dbo].[budget_month_prod] for insert, update,
delete
as

declare @bit int ,
        @field int ,
        @maxfield int ,
        @char int ,
        @fieldname varchar(128) ,
        @TableName varchar(128) ,
        @PKCols varchar(1000) ,
        @sql varchar(6000),
        @UpdateDate varchar(21) ,
        @UserName varchar(128) ,
        @Type char(1) ,
        @PKSelect varchar(1000),
        @PKfields varchar(1000) ,
        @PKvalues varchar(1000)
        DECLARE @FieldList TABLE(field VARCHAR(100));

        -- table name
        select @TableName = 'budget_month_prod'

        -- list of fields to be monitored
        INSERT INTO @FieldList VALUES('reviewerInput'),('contributorInput');


        -- date
        select  @UpdateDate = convert(varchar(8), getdate(), 112) + ' ' + convert(varchar(12),
getdate(), 114)

        -- Action and user
        if exists (select * from inserted)
        begin
                select  @UserName = (select distinct KP USER from inserted)

                if exists (select * from deleted)
                        select @Type = 'U'
                else
                        select @Type = 'I'
        end
        else
        begin
                select @Type = 'D'
                select  @UserName = (select distinct KP_USER from deleted)
        end

        if exists (SELECT * FROM INFORMATION SCHEMA.TABLES WHERE TABLE NAME = 'AUDIT ' + @TableName)
        begin
                -- get list of columns
                select * into #tins from inserted
```

```
                select * into #tdel from deleted

                -- Get primary key columns for full outer join
                select @PKCols = coalesce(@PKCols + ' and', ' on') + ' i.' + c.COLUMN_NAME + ' = d.'
+ c.COLUMN_NAME,
                                @PKfields = coalesce(@PKfields + ' ,', ' ') +  c.COLUMN_NAME
                from    INFORMATION_SCHEMA.TABLE_CONSTRAINTS pk , INFORMATION_SCHEMA.KEY_COLUMN_USAGE
c
                where   pk.TABLE_NAME = @TableName
                and     CONSTRAINT_TYPE = 'PRIMARY KEY'
                and     c.TABLE_NAME = pk.TABLE_NAME
                and     c.CONSTRAINT_NAME = pk.CONSTRAINT_NAME

                -- Get primary key select for insert
                select @PKvalues = coalesce(@PKvalues+','','') + ' convert(varchar(100),coalesce(i.' +
COLUMN_NAME +',d.' + COLUMN_NAME + '))'
                from    INFORMATION_SCHEMA.TABLE_CONSTRAINTS pk , INFORMATION_SCHEMA.KEY_COLUMN_USAGE
c
                where   pk.TABLE_NAME = @TableName
                and     CONSTRAINT_TYPE = 'PRIMARY KEY'
                and     c.TABLE_NAME = pk.TABLE_NAME
                and     c.CONSTRAINT_NAME = pk.CONSTRAINT_NAME

                select @field = 0, @maxfield = max(ORDINAL_POSITION) from INFORMATION_SCHEMA.COLUMNS
where TABLE_NAME = @TableName
                while @field < @maxfield
                begin
                        select @field = min(ORDINAL_POSITION) from INFORMATION_SCHEMA.COLUMNS where
TABLE_NAME = @TableName and ORDINAL_POSITION > @field
                        select @bit = (@field - 1 )% 8 + 1
                        select @bit = power(2,@bit - 1)
                        select @char = ((@field - 1) / 8) + 1
                        if substring(COLUMNS_UPDATED(),@char, 1) & @bit > 0 or @Type in ('I','D')
                        begin
                                select @fieldname = COLUMN_NAME from INFORMATION_SCHEMA.COLUMNS where
TABLE_NAME = @TableName and ORDINAL_POSITION = @field
                                select @sql =           'insert into AUDIT ' + @TableName + ' (Type, ' +
@PKfields + ', FieldName, OldValue, NewValue, UpdateDate, UserName)'
                                select @sql = @sql +   ' select ''' + @Type + ''''

                                select @sql = @sql +   ' , ' + @PKvalues + ''

                                select @sql = @sql +   ',''' + @fieldname + ''''
                                select @sql = @sql +   ',convert(varchar(1000),d.' + @fieldname + ')'
                                select @sql = @sql +   ',convert(varchar(1000),i.' + @fieldname + ')'
                                select @sql = @sql +   ',''' + @UpdateDate + ''''
                                select @sql = @sql +   ',''' + @UserName + ''''
                                select @sql = @sql +   ' from #tins i full outer join #tdel d'
                                select @sql = @sql +   @PKCols
                                select @sql = @sql +   ' where i.' + @fieldname + ' <> d.' + @fieldname
                                select @sql = @sql +   ' or (i.' + @fieldname + ' is null and  d.' +
@fieldname + ' is not null)'
                                select @sql = @sql +   ' or (i.' + @fieldname + ' is not null and  d.'
+ @fieldname + ' is null)'

                                if @fieldname in (SELECT field FROM @FieldList)
                                        exec (@sql)

                        end
                end
        end
        else
        begin
                if exists (SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME =
'AUDIT_DATABASE')
                begin

                        -- get list of columns
                        select * into #ins from inserted
                        select * into #del from deleted

                        -- Get primary key columns for full outer join
```

```
                        select  @PKCols = coalesce(@PKCols + ' and', ' on') + ' i.' + c.COLUMN_NAME +
' = d.' + c.COLUMN_NAME
                        from    INFORMATION_SCHEMA.TABLE_CONSTRAINTS pk ,
                                INFORMATION_SCHEMA.KEY_COLUMN_USAGE c
                        where   pk.TABLE_NAME = @TableName
                        and     CONSTRAINT_TYPE = 'PRIMARY KEY'
                        and     c.TABLE_NAME = pk.TABLE_NAME
                        and     c.CONSTRAINT_NAME = pk.CONSTRAINT_NAME

                        -- Get primary key select for insert
                        select @PKSelect = coalesce(@PKSelect+'+','') + '''' + COLUMN_NAME +
'=''+convert(varchar(100),coalesce(i.' + COLUMN_NAME +',d.' + COLUMN_NAME + '))+''';'''
                        from    INFORMATION_SCHEMA.TABLE_CONSTRAINTS pk ,
                                INFORMATION_SCHEMA.KEY_COLUMN_USAGE c
                        where   pk.TABLE_NAME = @TableName
                        and     CONSTRAINT_TYPE = 'PRIMARY KEY'
                        and     c.TABLE_NAME = pk.TABLE_NAME
                        and     c.CONSTRAINT_NAME = pk.CONSTRAINT_NAME

                        select @field = 0, @maxfield = max(ORDINAL_POSITION) from
INFORMATION_SCHEMA.COLUMNS where TABLE_NAME = @TableName
                        while @field < @maxfield
                        begin
                                select @field = min(ORDINAL_POSITION) from INFORMATION_SCHEMA.COLUMNS
where TABLE_NAME = @TableName and ORDINAL_POSITION > @field
                                select @bit = (@field - 1 )% 8 + 1
                                select @bit = power(2,@bit - 1)
                                select @char = ((@field - 1) / 8) + 1
                                if substring(COLUMNS_UPDATED(),@char, 1) & @bit > 0 or @Type in
('I','D')
                                begin
                                        select @fieldname = COLUMN_NAME from INFORMATION_SCHEMA.COLUMNS
where TABLE_NAME = @TableName and ORDINAL_POSITION = @field
                                        select @sql =           'insert into AUDIT_DATABASE (Type,
TableName, PK, FieldName, OldValue, NewValue, UpdateDate, UserName)'
                                        select @sql = @sql +    ' select ''' + @Type + ''''
                                        select @sql = @sql +    ',''' + @TableName + ''''
                                        select @sql = @sql +    ',' + @PKSelect
                                        select @sql = @sql +    ',''' + @fieldname + ''''
                                        select @sql = @sql +    ',convert(varchar(1000),d.' + @fieldname
+ ')'
                                        select @sql = @sql +    ',convert(varchar(1000),i.' + @fieldname
+ ')'
                                        select @sql = @sql +    ',''' + @UpdateDate + ''''
                                        select @sql = @sql +    ',''' + @UserName + ''''
                                        select @sql = @sql +    ' from #ins i full outer join #del d'
                                        select @sql = @sql +    @PKCols
                                        select @sql = @sql +    ' where i.' + @fieldname + ' <> d.' +
@fieldname
                                        select @sql = @sql +    ' or (i.' + @fieldname + ' is null and
d.' + @fieldname + ' is not null)'
                                        select @sql = @sql +    ' or (i.' + @fieldname + ' is not null
and  d.' + @fieldname + ' is null)'

                                        if @fieldname in (SELECT field FROM @FieldList)
                                                exec (@sql)
                                end
                        end
                end

        end

GO
```