Historisation des données

Partie 1 : mode colonne

par Frédéric Brouard, alias SQLpro
MVP SQL Server
Expert langage SQL, SGBDR, modélisation de données

Auteur de :
- SQLpro http://sqlpro.developpez.com/
- "SQL", coll. Synthex, avec C. Soutou, Pearson Education 2005
- "SQL" coll. Développement, Campus Press 2001

Enseignant aux Arts & Métiers et à l'ISEN Toulon

Alors que la version 2008 de Microsoft SQL Server offre un moyen de réaliser une historisation des données, intéressons nous aux différentes façons de faire cela à partir de la version 2005.

L'historisation des données, aussi appelé audit de données (audit trailing par exemple) est le mécanisme par lequel on capture les données à chaque évolution de ces dernières : modifications des valeurs des colonnes et suppressions des lignes. Cet outil est bien souvent nécessaire pour des raisons d'archivages de données anciennes de suivi de modifications ou encore pour tracer l'activité d'une base de données.
Cette série d'articles présente différentes méthodes assorties d'exemples.

1 - Introduction

L'historisation peut se faire pour collecter les changements de données sous trois formes différentes : en mode ligne, en mode colonne, en mode transactionnel.

En mode ligne il s'agit de stocker les lignes modifiées ou supprimées d'une table dans une table de même structure en y ajoutant certaines informations, comme la date/heure de l'événement de modification ou l'utilisateur qui a entrepris la mise à jour.

En mode colonne, il s'agit de stocker chacune des valeurs atomiques modifiées ou toutes les valeurs atomiques de toutes les lignes supprimées. De la même façon que précédemment on collectera des méta données de modification : qui, quand...

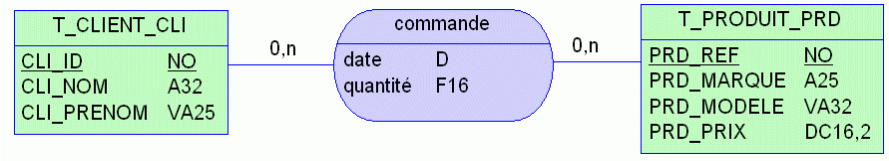
En mode transactionnel il s'agit de capturer, non pas les données, mais l'ordre SQL de modification. On y ajoutera de même les données que l'on jugera nécessaire pour le traitement. Il conviendra préalablement d'associer à ce mode, une sauvegarde de la base de données (cliché) à l'instant d'avant le démarrage du mécanisme d'historisation.

Le mode ligne est à préférer lorsqu'il y a beaucoup de suppression. Le mode colonne est à préférer lorsqu'il y a beaucoup de modifications (UPDATE). Le mode transactionnel est intéressant lorsque la base source doit être performante et que le SGBDR permet de mettre en oeuvre un tel mécanisme !

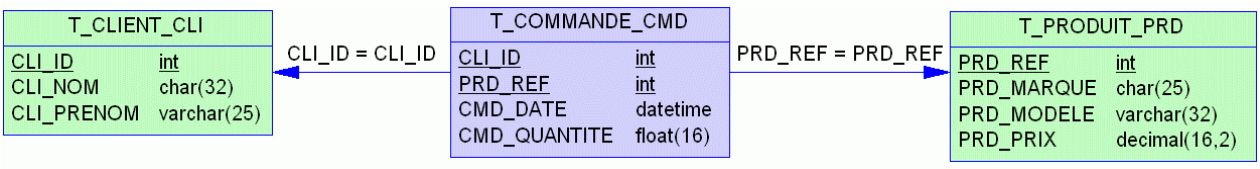
La présente série d'article étudie ces trois modes avec un exemple concret à l'aide de MS SQL Server 2005. Le présent article parle de l'historisation en mode ligne.

2 - La base exemple

La base exemple qui nous servira de fil rouge pour étudier ces différents mécanismes d'historisation est la suivante :



Modèle conceptuel de données



```
USE master;
GO

CREATE DATABASE DB_PROD;
GO

USE DB_PROD;
GO

/*==============================================================*/
/* Table : T_CLIENT_CLI                                         */
/*==============================================================*/
create table T_CLIENT_CLI (
   CLI_ID               int                  identity,
   CLI_NOM              char(32)             not null,
   CLI_PRENOM           varchar(25)          null,
   constraint PK_T_CLIENT_CLI primary key  (CLI_ID)
)
go

/*==============================================================*/
/* Table : T_COMMANDE_CMD                                       */
/*==============================================================*/
create table T_COMMANDE_CMD (
   CLI_ID               int                  not null,
   PRD_REF              int                  not null,
   CMD_DATE             datetime             not null,
   CMD_QUANTITE         float(16)            not null,
   constraint PK_T_COMMANDE_CMD primary key  (CLI_ID, PRD_REF)
)
go

/*==============================================================*/
/* Index : T_COMMANDE_CMD2_FK                                   */
/*==============================================================*/
create   index T_COMMANDE_CMD2_FK on T_COMMANDE_CMD (
PRD_REF
)
go

/*==============================================================*/
/* Table : T_PRODUIT_PRD                                        */
/*==============================================================*/
create table T_PRODUIT_PRD (
   PRD_REF              int                  identity,
   PRD_MARQUE           char(25)             not null,
   PRD_MODELE           varchar(32)          not null,
   PRD_PRIX             decimal(16,2)        not null,
   constraint PK_T_PRODUIT_PRD primary key  (PRD_REF)
)
go

alter table T_COMMANDE_CMD
   add constraint FK_T_COMMAN_T_COMMAND_T_CLIENT foreign key (CLI_ID)
      references T_CLIENT_CLI (CLI_ID)
go

alter table T_COMMANDE_CMD
   add constraint FK_T_COMMAN_T_COMMAND_T_PRODUI foreign key (PRD_REF)
      references T_PRODUIT_PRD (PRD_REF)
go
```

script sql de création des objets de la base

Pour ce qui est des données historisée, nous avons décidé pour illustrer les différents concepts de créer trois bases de données : DB_HST_LIGNE, DB_HST_COL, DB_HST_SQL :

```
USE master;
GO

CREATE DATABASE DB_HST_LIGNE;
GO
CREATE DATABASE DB_HST_COL;
GO
CREATE DATABASE DB_HST_SQL;
GO
```

script de création des bases de données d'historisation

NOTA : dans cet exemple, nous avons respecté notre norme de nommage des noms des objets qui veut que toute table soit suffixée par un trigramme unique et que toute colonne d'une table (sauf clef étrangères) reprenne en préfixe le trigramme de la table. Sans le respect de ces éléments il convient de modifier le code donné en exemple.

3 - Historisation en mode ligne

Il convient de créer une table d'historisation pour chaque table dont on veut suivre les données. Chaque table d'historisation contient exactement les mêmes définitions de colonnes que la table dont on suit les évolutions de données et peut contenir en sus, les colonnes suivantes :

ID	colonne auto incrémentée servant de clef
MD	mode de mise à jour (U : update, D : delete)
DH	date et heure de modification (système)
SU	"SQL user", c'est à dire utilisateur SQL
MA	Mac Adress (du PC ayant demandé la modification).

Cette liste n'étant pas limitative.
Afin de distinguer les noms des colonnes correspondant à ces attributs, nous allons préfixer leur nom par un blanc souligné dans la table d'historisation.
Notons qu'il est nécessaire de s'affranchir de toute contrainte de la table d'origine. Par exemple les contraintes NOT NULL, PRIMARY KEY, FOREIGN KEY, CHECK, UNIQUE n'on pas à figurer dans la table d'historisation.

Si votre SGBDR permet de créer des déclencheurs DDL alors il est facile de capturer un événement de création de table et de créer dans la foulée une table d'historisation.

Exemple pour SQL Server 2005 :

```
USE DB_PROD;
GO

CREATE TRIGGER E_DB_CRETAB
ON DATABASE
FOR CREATE_TABLE
AS
BEGIN

SET NOCOUNT ON;

-- récupération des informations du "paquet" d'événement du tgrigger DDL
DECLARE @XML XML, @SCH sysname, @TAB sysname;
SET @XML = EVENTDATA();
```

```
-- extraction à l'aide d'XQuery/XPath du nom du schema et du nom de table
SELECT @SCH = @XML.value('(/EVENT_INSTANCE/SchemaName)[1]', 'sysname'),
       @TAB = @XML.value('(/EVENT_INSTANCE/ObjectName)[1]', 'sysname');

-- génération d'une requête de création de la table d'historisation

DECLARE @SQL VARCHAR(max);
-- un schéma existe-il avec ce nom là ?
IF NOT EXISTS (SELECT *
               FROM   DB_HST_LIGNE.INFORMATION_SCHEMA.SCHEMATA
               WHERE  SCHEMA_NAME = @SCH)
BEGIN
-- non : on le créé
   SET @SQL = 'CREATE SCHEMA ' + @SCH;
   EXEC (@SQL);
END;

-- création de la table
SET @SQL = 'CREATE TABLE DB_HST_LIGNE.' + @SCH+'.'+@TAB + ' ('
    + '_ID BIGINT NOT NULL IDENTITY PRIMARY KEY, _MD CHAR(1), '
    + '_DH DATETIME DEFAULT CURRENT_TIMESTAMP, '
    + '_SU NVARCHAR(128) DEFAULT USER, _MA NCHAR(40), ';
SELECT @SQL = @SQL + COLUMN_NAME + ' ' + DATA_TYPE +
       CASE
          WHEN DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar')
             THEN ' (' + CAST(CHARACTER_MAXIMUM_LENGTH AS VARCHAR(16))
                  + ') COLLATE ' + COLLATION_NAME
          WHEN DATA_TYPE IN ('decimal', 'numeric')
             THEN ' (' + CAST(NUMERIC_PRECISION AS VARCHAR(16)) +', '
                  + CAST(NUMERIC_SCALE AS VARCHAR(16)) + ')'
          ELSE ''
       END + ', '
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  TABLE_SCHEMA = @SCH
  AND  TABLE_NAME = @TAB;
SET @SQL = SUBSTRING(@SQL, 1, LEN(@SQL) - 1) + ')';
EXEC (@SQL);

END;
```

déclencheur DDL MS SQL Server 2005 pour capture d'un CREATE TABLE
et composition d'une table clone d'historisation

On devra faire de même avec des déclencheurs DDL ALTER TABLE afin de répercuter les évolutions du schéma de la table. Mais là, notre affaire se complique. En effet, étudions les différents cas de figure et leurs solutions :

Ordre SQL	Préserver les informations	Ne pas préserver les informations
ALTER TABLE ... DROP COLUMN ...	Ne pas répercuter la modification dans la table d'historisation	Répercuter la modification dans la table d'historisation
ALTER TABLE ... ADD COLUMN ...	Répercuter la modification dans la table d'historisation sans tenir compte des contraintes	Répercuter la modification dans la table d'historisation
ALTER TABLE ... ALTER COLUMN ...	Répercuter la modification dans la table d'historisation sans tenir compte des contraintes	Répercuter la modification dans la table d'historisation

Un tel déclencheur pourrait s'écrire :

```
USE DB_PROD;
GO

CREATE TRIGGER E_DB_ALTTAB
ON DATABASE
FOR ALTER_TABLE
AS
```

```
BEGIN

SET NOCOUNT ON;

-- récupération des informations du "paquet" d'événement du tgrigger DDL
DECLARE @XML XML, @SCH sysname, @TAB sysname;
SET @XML = EVENTDATA();

-- extraction à l'aide d'XQuery/XPath du nom du schema et du nom de table
SELECT @SCH = @XML.value('(/EVENT_INSTANCE/SchemaName)[1]', 'sysname'),
       @TAB = @XML.value('(/EVENT_INSTANCE/ObjectName)[1]', 'sysname');

-- génération d'une requête de création de la table d'historisation

DECLARE @SQL VARCHAR(max);

/*
-- cette modification a t-elle ajoutée des colonnes de la table ?
-- supression d'une colonne => la colonne est renommée en #001, #002, etc...
-- changement de type d'une colonne => l'ancienne colonne est renomée _#001 et la
nouvelle ajoutée
-- ajout d'une colonne,
*/

-- c'est une modification de type, voici comment on la détecte :
SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME,
NUMERIC_PRECISION, NUMERIC_SCALE
FROM   DB_HST_LIGNE.INFORMATION_SCHEMA.COLUMNS AS T
       INNER JOIN (SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                   FROM   DB_HST_LIGNE.INFORMATION_SCHEMA.COLUMNS
                   WHERE  TABLE_SCHEMA = @SCH
                     AND  TABLE_NAME = @TAB
                   EXCEPT
                   SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                   FROM   INFORMATION_SCHEMA.COLUMNS
                   WHERE  TABLE_SCHEMA = @SCH
                     AND  TABLE_NAME = @TAB) AS TE
            ON T.TABLE_SCHEMA = TE.TABLE_SCHEMA
               AND T.TABLE_NAME = TE.TABLE_NAME
               AND T.COLUMN_NAME = TE.COLUMN_NAME
WHERE  NOT EXISTS(SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME--, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                  FROM   DB_HST_LIGNE.INFORMATION_SCHEMA.COLUMNS
                  WHERE  TABLE_SCHEMA = @SCH
                    AND  TABLE_NAME = @TAB
                  EXCEPT
                  SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME--, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                  FROM   INFORMATION_SCHEMA.COLUMNS
                  WHERE  TABLE_SCHEMA = @SCH
                    AND  TABLE_NAME = @TAB
                  UNION
                  SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME--, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                  FROM   INFORMATION_SCHEMA.COLUMNS
                  WHERE  TABLE_SCHEMA = @SCH
                     AND  TABLE_NAME = @TAB
                  EXCEPT
                  SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME--, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                  FROM   DB_HST_LIGNE.INFORMATION_SCHEMA.COLUMNS
                  WHERE  TABLE_SCHEMA = @SCH
                    AND  TABLE_NAME = @TAB)
IF @@ROWCOUNT > 0
BEGIN
### faire le boulot !!!
END

-- c'est un ajout, voici comment on le détecte :
SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME,
NUMERIC_PRECISION, NUMERIC_SCALE
FROM   INFORMATION_SCHEMA.COLUMNS AS T
       INNER JOIN (SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME--, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                   FROM   INFORMATION_SCHEMA.COLUMNS
```

```
                    WHERE  TABLE_SCHEMA = @SCH
                      AND  TABLE_NAME = @TAB
                    EXCEPT
                    SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME--, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                    FROM   DB_HST_LIGNE.INFORMATION_SCHEMA.COLUMNS
                    WHERE  TABLE_SCHEMA = @SCH
                      AND  TABLE_NAME = @TAB) AS TE
             ON T.TABLE_SCHEMA = TE.TABLE_SCHEMA
                AND T.TABLE_NAME = TE.TABLE_NAME
                AND T.COLUMN_NAME = TE.COLUMN_NAME
IF @@ROWCOUNT > 0
BEGIN
### faire le boulot !!!
END


-- c'est une suppression, voici comment on la détecte :
SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME,
NUMERIC_PRECISION, NUMERIC_SCALE
FROM   DB_HST_LIGNE.INFORMATION_SCHEMA.COLUMNS AS T
       INNER JOIN (SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME--, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                    FROM   DB_HST_LIGNE.INFORMATION_SCHEMA.COLUMNS
                    WHERE  TABLE_SCHEMA = @SCH
                      AND  TABLE_NAME = @TAB
                    EXCEPT
                    SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME--, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_SCALE
                    FROM   INFORMATION_SCHEMA.COLUMNS
                    WHERE  TABLE_SCHEMA = @SCH
                      AND  TABLE_NAME = @TAB) AS TE
             ON T.TABLE_SCHEMA = TE.TABLE_SCHEMA
                AND T.TABLE_NAME = TE.TABLE_NAME
                AND T.COLUMN_NAME = TE.COLUMN_NAME
IF @@ROWCOUNT > 0
BEGIN
### faire le boulot !!!
END


### modification à reprendre :
-- il faut ensuite modifier la table d'historisation suivant les différents cas de
figure :
SET @SQL = 'ALTER TABLE DB_HST_LIGNE.' + @SCH+'.'+@TAB + ' ADD ###';
SELECT @SQL = @SQL + COLUMN_NAME + ' ' + DATA_TYPE +
       CASE
          WHEN DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar')
             THEN ' (' + CAST(CHARACTER_MAXIMUM_LENGTH AS VARCHAR(16))
                  + ') COLLATE ' + COLLATION_NAME
          WHEN DATA_TYPE IN ('decimal', 'numeric')
             THEN ' (' + CAST(NUMERIC_PRECISION AS VARCHAR(16)) +', '
                  + CAST(NUMERIC_SCALE AS VARCHAR(16)) + ')'
          ELSE ''
       END + ', '
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  TABLE_SCHEMA = @SCH
  AND  TABLE_NAME = @TAB;
SET @SQL = SUBSTRING(@SQL, 1, LEN(@SQL) - 1) + ')';
EXEC (@SQL);

END;
```

Vous noterez que nous avons laissé en commentaire la partie de code qui doit réaliser la modification de la table d'historisation. Le problème n'est pas trivial mais le code serait trop long à présenter dans cet article

Il va falloir maintenant implanter le jeu de déclencheurs permettant de capturer les INSERT et les UPDATE. Ces déclencheurs peuvent eux aussi être réalisés dans le déclencheur DDL.

3.1 - Déclencheur pour capture des UPDATE

Il devra revêtir la forme :

```
CREATE TRIGGER E_D_CLI
ON dbo.T_CLIENT_CLI
FOR DELETE
AS
BEGIN

   INSERT INTO DB_HST_LIGNE.dbo.T_CLIENT_CLI
               (_MA, _MD, CLI_ID, CLI_NOM, CLI_PRENOM)
   SELECT client_net_address, 'D', d.*
   FROM   deleted d
          CROSS JOIN sys.dm_exec_connections
   WHERE  session_id = @@SPID

END
```

On peut, bien entendu le créer dynamiquement dans la foulée de la table d'historisation dans le déclencheur DDL :

```
-- création du déclencheur DDL
SET @SQL = 'CREATE TRIGGER E_U_' + SUBSTRING(@TAB, LEN(@TAB) -2, 3) +
           ' ON ' + @SCH + '.' + @TAB +
           ' FOR UPDATE AS BEGIN SET NOCOUNT ON INSERT INTO DB_HST_LIGNE.'
          + @SCH + '.' + @TAB + ' (_MA, _MD, ';
SELECT @SQL = @SQL +  COLUMN_NAME + ', '
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  TABLE_SCHEMA = @SCH
  AND  TABLE_NAME = @TAB;
SET @SQL = SUBSTRING(@SQL, 1, LEN(@SQL) - 1) + ') '+
           'SELECT client_net_address, ''U'', i.* ' +
           'FROM   inserted i ' +
           'CROSS JOIN sys.dm_exec_connections ' +
           'WHERE  session_id = @@SPID ' +
           'END '
EXEC (@SQL);
```

Code à rajouter au déclencheur DDL afin de créer le déclencheur DML UPDATE pour suivi d'historisation.

3.2 - Déclencheur pour capture des DELETE

Il n'est guère plus compliqué que le précédent :

```
CREATE TRIGGER dbo.E_D_CLI
ON dbo.T_CLIENT_CLI
FOR DELETE
AS
BEGIN

   INSERT INTO DB_HST_LIGNE.dbo.T_CLIENT_CLI (_MA, _MD, CLI_ID, CLI_NOM,
CLI_PRENOM)
   SELECT client_net_address, 'D', d.*
   FROM   deleted d
          CROSS JOIN sys.dm_exec_connections
   WHERE  session_id = @@SPID

END
```

Et peut, comme précédemment, être créé dynamiquement dans la foulée de la table d'historisation dans le déclencheur DDL :

```
-- création du déclencheur DDL DELETE de suivi de suppression
SET @SQL = 'CREATE TRIGGER E_D_' + SUBSTRING(@TAB, LEN(@TAB) -2, 3) +
           ' ON ' + @SCH + '.' + @TAB +
           ' FOR DELETE AS SET NOCOUNT ON BEGIN INSERT INTO DB_HST_LIGNE.'
          + @SCH + '.' + @TAB + ' (_MA, _MD, ';
SELECT @SQL = @SQL +  COLUMN_NAME + ', '
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  TABLE_SCHEMA = @SCH
  AND  TABLE_NAME = @TAB;
SET @SQL = SUBSTRING(@SQL, 1, LEN(@SQL) - 1) + ') '+
           'SELECT client_net_address, ''D'', d.* ' +
           'FROM   deleted d ' +
           'CROSS JOIN sys.dm_exec_connections ' +
           'WHERE  session_id = @@SPID ' +
           'END '
EXEC (@SQL);
```

Code à rajouter au déclencheur DDL afin de créer le déclencheur DML DELETE pour suivi d'historisation.

Dans le cas ou l'on aurait choisit de ne pas tenir compte des évolutions du schéma des tables traquées, alors il conviendrait de modifier les déclencheurs afin que soit spécifié de manière exhaustive les colonnes cibles dans les insertions des tables d'historisation.

3.3 - Déclencheur DDL complet

Le code complet du déclencheur DDL est donc le suivant :

```
CREATE TRIGGER E_DB_CRETAB
ON DATABASE
FOR CREATE_TABLE
AS
BEGIN

SET NOCOUNT ON;

-- récupération des informations du "paquet" d'événement du trigger DDL
DECLARE @XML XML, @SCH sysname, @TAB sysname;
SET @XML = EVENTDATA();

-- extraction à l'aide d'XQuery/XPath du nom du schema et du nom de table
SELECT @SCH = @XML.value('(/EVENT_INSTANCE/SchemaName)[1]', 'sysname'),
       @TAB = @XML.value('(/EVENT_INSTANCE/ObjectName)[1]', 'sysname');

-- génération d'une requête de création de la table d'historisation
DECLARE @SQL VARCHAR(max);
-- un schéma existe-il avec ce nom là ?
IF NOT EXISTS (SELECT *
               FROM   DB_HST_LIGNE.INFORMATION_SCHEMA.SCHEMATA
               WHERE  SCHEMA_NAME = @SCH)
BEGIN
-- non : on le créé
   SET @SQL = 'CREATE SCHEMA ' + @SCH;
   EXEC (@SQL);
END;

-- création de la table
SET @SQL = 'CREATE TABLE DB_HST_LIGNE.' + @SCH+'.'+@TAB + ' ('
    + '_ID BIGINT NOT NULL IDENTITY PRIMARY KEY, _MD CHAR(1), '
    + '_DH DATETIME DEFAULT CURRENT_TIMESTAMP, '
    + '_SU NVARCHAR(128) DEFAULT USER, _MA NCHAR(40), ';
SELECT @SQL = @SQL + COLUMN_NAME + ' ' + DATA_TYPE +
       CASE
          WHEN DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar')
             THEN ' (' + CAST(CHARACTER_MAXIMUM_LENGTH AS VARCHAR(16))
                  + ') COLLATE ' + COLLATION_NAME
```

```
          WHEN DATA_TYPE IN ('decimal', 'numeric')
             THEN ' (' + CAST(NUMERIC_PRECISION AS VARCHAR(16)) +', '
                  + CAST(NUMERIC_SCALE AS VARCHAR(16)) + ')'
          ELSE ''
       END + ', '
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  TABLE_SCHEMA = @SCH
  AND  TABLE_NAME = @TAB;
SET @SQL = SUBSTRING(@SQL, 1, LEN(@SQL) - 1) + ')';
EXEC (@SQL);


-- création du déclencheur DDL UPDATE de suivi de modification
SET @SQL = 'CREATE TRIGGER E_U_' + SUBSTRING(@TAB, LEN(@TAB) -2, 3) +
           ' ON ' + @SCH + '.' + @TAB +
           ' FOR UPDATE AS SET NOCOUNT ON BEGIN INSERT INTO DB_HST_LIGNE.'
+
           @SCH + '.' + @TAB + ' (_MA, _MD, ';
SELECT @SQL = @SQL +  COLUMN_NAME + ', '
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  TABLE_SCHEMA = @SCH
  AND  TABLE_NAME = @TAB;
SET @SQL = SUBSTRING(@SQL, 1, LEN(@SQL) - 1) + ') '+
           'SELECT client_net_address, ''U'', i.* ' +
           'FROM   inserted i ' +
           'CROSS JOIN sys.dm_exec_connections ' +
           'WHERE  session_id = @@SPID ' +
           'END '
EXEC (@SQL);


-- création du déclencheur DDL DELETE de suivi de suppression
SET @SQL = 'CREATE TRIGGER E_D_' + SUBSTRING(@TAB, LEN(@TAB) -2, 3) +
           ' ON ' + @SCH + '.' + @TAB +
           ' FOR DELETE AS SET NOCOUNT ON BEGIN INSERT INTO DB_HST_LIGNE.'
+
           @SCH + '.' + @TAB + ' (_MA, _MD, ';
SELECT @SQL = @SQL +  COLUMN_NAME + ', '
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  TABLE_SCHEMA = @SCH
  AND  TABLE_NAME = @TAB;
SET @SQL = SUBSTRING(@SQL, 1, LEN(@SQL) - 1) + ') '+
           'SELECT client_net_address, ''D'', d.* ' +
           'FROM   deleted d ' +
           'CROSS JOIN sys.dm_exec_connections ' +
           'WHERE  session_id = @@SPID ' +
           'END '
EXEC (@SQL);


END;
```

Déclencheur DDL s'occupant de créer la table d'historisation par clonage et les déclencheurs DML UPDATE et DELETE pour suivi de modification.

3.4 Test de la solution

On pourra tester cette solution avec les données suivantes :

```
USE DB_PROD;
GO

INSERT INTO dbo.T_CLIENT_CLI VALUES ('Duchemin', 'Marcel');
INSERT INTO dbo.T_CLIENT_CLI VALUES ('Montel', 'Marc');
GO

UPDATE dbo.T_CLIENT_CLI
SET    CLI_NOM = UPPER(CLI_NOM);
```

```
GO

DELETE FROM dbo.T_CLIENT_CLI
WHERE  CLI_PRENOM = 'marc' COLLATE French_CI_AI;
GO

SELECT *
FROM   DB_HST_LIGNE.dbo.T_CLIENT_CLI
UNION  ALL
SELECT NULL, NULL, NULL, NULL, NULL, *
FROM   dbo.T_CLIENT_CLI
ORDER  BY 6, 1 DESC
```

Script de test pour suivi des mises à jour

La dernière requête devant conduire à présenter les données de la sorte :

	_ID	_MD	_DH	_SU	_MA	CLI_ID	CLI_NOM	CLI_PRENOM
1	2	U	2008-06-20 20:57:34.640	dbo	<local machine>	1	DUCHEMIN	Marcel
2	NULL	NULL	NULL	NULL	NULL	1	DUCHEMIN	Marcel
3	3	D	2008-06-20 20:57:34.700	dbo	<local machine>	2	MONTEL	Marc
4	1	U	2008-06-20 20:57:34.640	dbo	<local machine>	2	MONTEL	Marc

on l'on voir que la seule ligne encore existante de la table originelle est la première du lot (cinq premières colonnes à NULL).