Les tables d'historiques avec SQL 2016

Qu'est-ce que c'est?

Leur dénomination en anglais est "System-Versioned Temporal Tables". En français, on dira "Tables Temporelles avec Contrôle de Version par le Système". En abrégé, on les appelle les tables d'historiques, ou tables temporelles.
Ce sont des tables qui conservent de façon automatique, permanente et horodatée les différentes versions successives de chaque ligne créée, modifiée, ou supprimée.

Vous pouvez selon vos besoins démarrer ce service en créant une nouvelle table, ou bien à partir d'une table existant déjà.
Dans les deux cas, le serveur SQL créera une deuxième table pour enregistrer les versions de l'historique.

On appellera table principale celle qui contient les données actuelles, que vous créez/modifiez/supprimez selon les besoins de vos applications. C'est le moteur SQL qui va alimenter la table d'historique liée, sans que vous ayez quoi que ce soit à changer à votre code existant.
L'ensemble des 2 tables forme la table temporelle.

À quoi ça sert?

- Si vous avez besoin de prouver que les données d'une table n'ont pas été modifées en dehors de votre application
- Si vous devez auditer les états successifs qui ont conduit à conserver les données telles qu'elles sont
- Si vous avez besoin de "restaurer" une seule table, par exemple pour corriger une serie d'erreurs de saisie
- Si vous cherchez les ruptures et anomalies dans des données métier
- Etc, etc...

Quels sont les prérequis et les restrictions?

Vous devez utiliser SQL 2016, quelle que soit la version. Les tables dhistoriques fonctionnent parfaitement avec SQL Express, et même avec LocalDB.

- La table principale doit posséder une clé primaire,
- et devra contenir deux colonnes dites "de période", de type DateTime2 qui seront utilisées par le système.
  Une fois installées, vous n'aurez plus à vous préoccuper de ces deux colonnes qui ne seront jamais utilisées dans votre code; toute référence à ces colonnes dans un INSERT ou un UPDATE lèvera une erreur.
- Il est impossible de faire un TRUNCATE TABLE sur une table temporelle
- Les triggers INSTEAD OF ne sont pas autorisés (ils casseraient la logique de traçage temporel)
- Une table temporelle ne peut pas être référencée comme cible dans un ON DELETE CASCADE ou un ON UPDATE CASCADE (mais vous pouvez utiliser des triggers AFTER pour gérer l'intégrité référentielle dans ce cas)
- Les colonnes de type FileStream ne sont pas permises, mais vous pouvez utiliser les données blob incluses telles que varchar(max), nvarchar(max), varbinary(max)
- Les tables de type FileTable ne peuvent pas être temporalisées

Allons-y...

Nous utiliserons l'Explorateur de Données de VFP9 pour créer les tables sur le serveur SQL, mais libre à vous d'adapter ce code dans des vues distantes, du SQL pass-through, ou des cursor adapters.
Tous les exemples ci-dessous sont exécutés sur une database de test, nommée TestHisto, dont on assumera

qu'elle est déjà créée sur l'instance SQL Express.

Nous utiliserons des vues distantes pour requêter les tables construites, mais vous pouvez utiliser également les CA ou du SQL Pass-Through.

Création d'une table temporelle

Voici la syntaxe la plus simple pour créer une table temporelle (table principale + table d'historique) en une seule commande:

```
CREATE TABLE Essaihisto1

   (Pk_Histo1    INT IDENTITY PRIMARY KEY,
    Champ1       VARCHAR(50) NOT NULL,
    Champ2       INT NULL,
    Champ3       BIT NULL,
    Horodatedebut DATETIME2 GENERATED ALWAYS AS ROW START NOT NULL,
    Horodatefin      DATETIME2 GENERATED ALWAYS AS ROW END     NOT NULL,
    PERIOD FOR SYSTEM_TIME(Horodatedebut, Horodatefin)
   )
         WITH(SYSTEM_VERSIONING = ON);
```

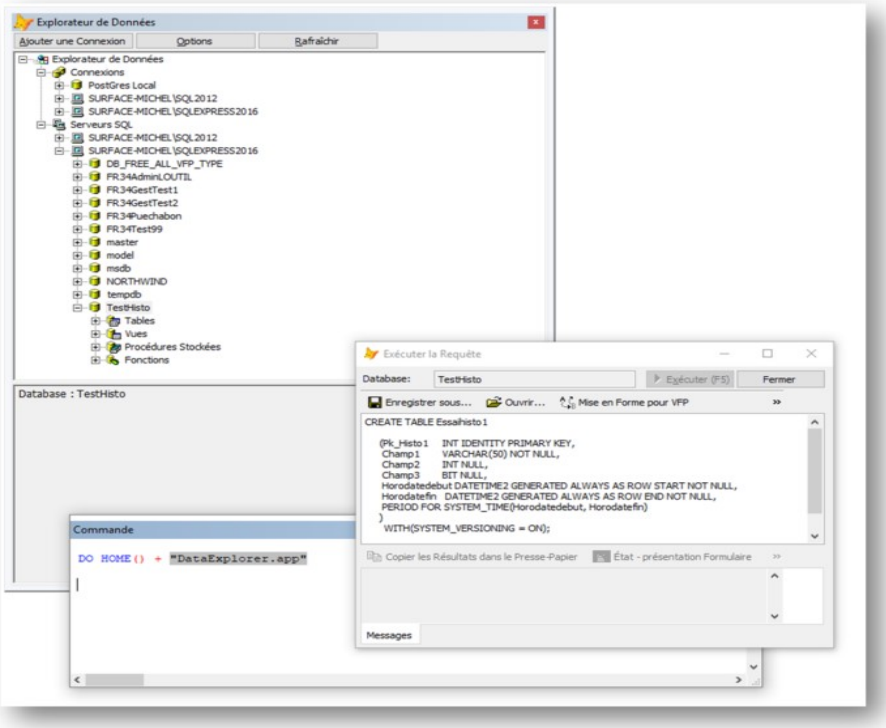
Explications sur la syntaxe minimum

- Vous pouvez nommer vos deux colonnes d'horodatage comme vous le voulez, elles doivent seulement être de type DATETIME2
- Vous devez indiquer pour ces deux colonnes l'argument GENERATED ALWAYS AS ROW START NOT NULL et GENERATED ALWAYS AS ROW END NOT NULL
- Vous devez les mentionner comme PERIOD FOR SYSTEM_TIME (<colonne de début>,<colonne de fin>)
- Et vous demandez la création de la table d'historique avec le WITH(SYSTEM_VERSIONING = ON)

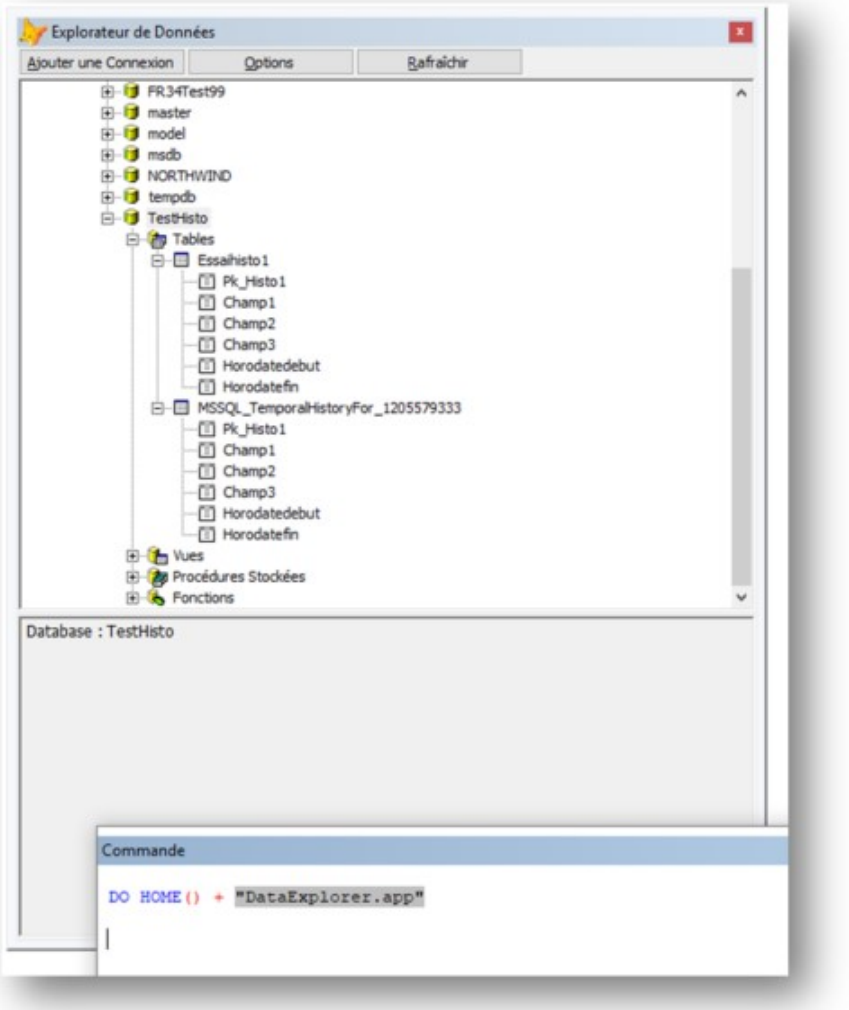
Exécution dans VFP

Création de la table

Lancez l'explorateur de données, cliquez droit sur le noeud représentant la database de test, choisissez "Exécuter la requête", et collez-y le code ci-dessus, puis Exécuter (F5).



Raffraichissez votre explorateur de données, et regardez maintenant les tables de la database: vous y voyez la table EssaiHisto1, mais aussi une table nommée MSSQL_TemporalHistoryFor_nnnnnnnnn



Cette deuxième table est la table d'historique.

Elle présente bien les mêmes colonnes que la table principale, mais sans aucune contrainte de clé primaire.

Insertion, modification, et suppression de lignes

Créez un dbc de test, ajoutez-lui une connexion vers le database SQL TestHisto, et créez une vue distante vers la table principale. Il est inutile de mettre les deux colonnes d'horodatage dans cette vue, puisqu'elles sont automatiquement valuées par le serveur. Nous verrons que nous ne les utiliserons pas non plus pour requêter l'historique (je vous montrerai ensuite comment les masquer)
Ajoutez des enregistrements, modifiez-en, supprimez-en. Tout fonctionne comme à l'ordinaire.

Interrogation de l'historique

L'interrogation de l'historique se fait depuis la table principale, en ajoutant l'argument **FOR SYSTEM_TIME** à votre SELECT.
Cet argument attend 4 clauses possibles, que nous détaillons maintenant:

Pour obtenir tout l'historique de votre table, utilisez la clause ALL

```
SELECT * FROM EssaiHisto1 FOR SYSTEM_TIME ALL;
```

Nous n'avons pas masqué les colonnes d'horodatage dans la définition de la table, nous pouvons donc voir ce qu'elles contiennent.
Dans l'explorateur de données, démarrez une fenêtre de requête et lancez le code suivant:

```
SELECT * FROM [dbo].[Essaihisto1];
SELECT * FROM [dbo].[Essaihisto1] FOR SYSTEM_TIME ALL;
```

Vous obtiendrez deux volets de résultat qui vous permettent de comprendre le fonctionnement interne des colonnes d'horodatage:

- quand une ligne est créée dans la table principale (INSERT), elle est ajoutée dans la table d'historique
  la colonne déclarée comme colonne de début dans le FOR SYSTEM TIME contient l'horodatage du début de la transaction d'insertion
  la colonne déclarée comme colonne de fin dans le FOR SYSTEM TIME n'est pas modifiée (31 décembre 9999...)
- quand une ligne est modifiée dans la table pricipale (UPDATE), la colonne déclarée comme colonne de fin dans le FOR SYSTEM TIME contient l'horodatage du début de la transaction de modification pour la ligne d'insertion si c'est le premier update de cette ligne dans la table principale, ou pour la dernière ligne de modification si on a déjà fait des update sur cette ligne en table principale.
- quand une ligne est supprimée dans la table principale (DELETE), la colonne déclarée comme colonne de fin dans le FOR SYSTEM TIME contient l'horodatage du début de la transaction de suppression pour la ligne d'insertion si on n'a jamais fait d'update sur cette ligne dans la table principale, ou pour la dernière ligne de modification si la ligne insérée a fait l'objet d'un update en table principale.

Pour obtenir une vue de votre table à une date-heure spécifique, utilisez la clause AS OF <votre date-heure>

```
SELECT * FROM EssaiHisto1 FOR SYSTEM_TIME AS OF  '2017-05-28 14:30';
```

Vous obtenez votre table telle qu'elle était le 28 mai 2017 à 14h30.

Pour obtenir l'historique des modifications de lignes spécifiques sur une période

Pour gérer les périodes (de telle date-heure à telle autre date-heure), vous disposez de 3 clauses possibles, selon le type d'historique que vous désirez et selon que vous voulez inclure ou exclure les bornes.

- si vous voulez les versions entièrement contenues dans une période bornes incluses, utilisez CONTAINED IN (<date-heure de début>, <date-heure de fin>)
  par exemple, je veux les versions du 28 mai 2017 entre 14h30 et 14h32.

```
SELECT * FROM EssaiHisto1 FOR SYSTEM_TIME CONTAINED IN (  '2017-05-28 14:30' , '2017-05-28 14:32);
```

- si vous voulez les versions qui ont commencé avant une date limite ou bien qui ont fini après une date plancher, utilisez FROM ou BETWEEN
- pour avoir les 2 bornes incluses [début, fin], utilisez FROM <date-heure de début> TO <date-heure de fin>
  par exemple, je veux les versions d'après 14h30 ou bien d'avant 14h32, ces deux bornes incluses

```
SELECT * FROM EssaiHisto1 FOR SYSTEM_TIME FROM  '2017-05-28 14:30' TO '2017-05-28 14:32;
```

- pour exclure la borne de fin [début, fin[, utilisez BETWEEN <date-heure de début> AND <date-heure de fin>
  par exemple, je veux les versions d'après 14h30 ou bien d'avant 14h32, 14h30 inclus mais pas 14h32

```
SELECT * FROM EssaiHisto1 FOR SYSTEM_TIME BETWEEN  '2017-05-28 14:30' AND '2017-05-28 14:32;
```

Comment masquer les colonnes d'horodatage

En masquant les colonnes d'horodatage, elles ne seront plus incluses dans le retour d'un SELECT *. Celà peut être utilise aussi dans le cas où vous transformez une table existant déjà en table versionnée par le système, et que vosu ne voulez pas modifer le code des applications clientes.

Le mot-clé à utiliser est HIDDEN

En création de table, vous ajoutez simplement HIDDEN dans la définition de colonne

```
CREATE TABLE Essaihisto1

   (Pk_Histo1    INT IDENTITY PRIMARY KEY,
    ...,
    Horodatedebut DATETIME2 GENERATED ALWAYS AS ROW START HIDDEN NOT NULL,
    Horodatefin      DATETIME2 GENERATED ALWAYS AS ROW END    HIDDEN NOT NULL,
    PERIOD FOR SYSTEM_TIME(Horodatedebut, Horodatefin)
   )
         WITH(SYSTEM_VERSIONING = ON);
```

en modification vous le mettez dans votre ALTER TABLE

```
ALTER TABLE … ALTER COLUMN … ADD HIDDEN
```

pour rétablir la visibilité, utilisez la même syntaxe avec un DROP HIDDEN

Comment purger l'historique

Rappelons que la table d'historique est en lecture seule, on ne peut donc y faire aucun DELETE ni TRUNCATE, pas plus qu'on ne peut demander de DROP. Les données sont compressées, certes, mais enfin il faudra bien à un moment ou un autre purger cet historique si on ne veut pas bloquer le disque.

Pour pouvoir supprimer des lignes dans la table d'historique, il faut impérativement la "débrancher" de la table principale avec un ALTER TABLE <table principale> SET (SYSTEM_VERSIONING = OFF), puis faire un DELETE sur la condition voulue, et enfin "rebrancher" la table d'historique.

La plupart du temps, la condition de DELETE portera sur la date de début, mais ce peut être toute condition métier de votre choix, ou toute obligation légale nouvellement introduite.

Il y a deux points qui nécessitent une attention particulière:

- pour rebrancher la table d'historique existante après l'avoir purgée, il faut absolument donner le nom de celle-ci dans l'instruction SET (SYSTEM_VERSIONING = ON, sinon SQL Server va en recréer une nouvelle, qui sera vide, et la continuité de l'historique sera cassée.
  Il est donc plus simple de nommer nous-même la table d'historique lors de la création, en ajoutant l'argument HISTORY_TABLE = <nom de la table d'historique>. Attention! il faut obligatoirement que le nom du schéma soit présent!

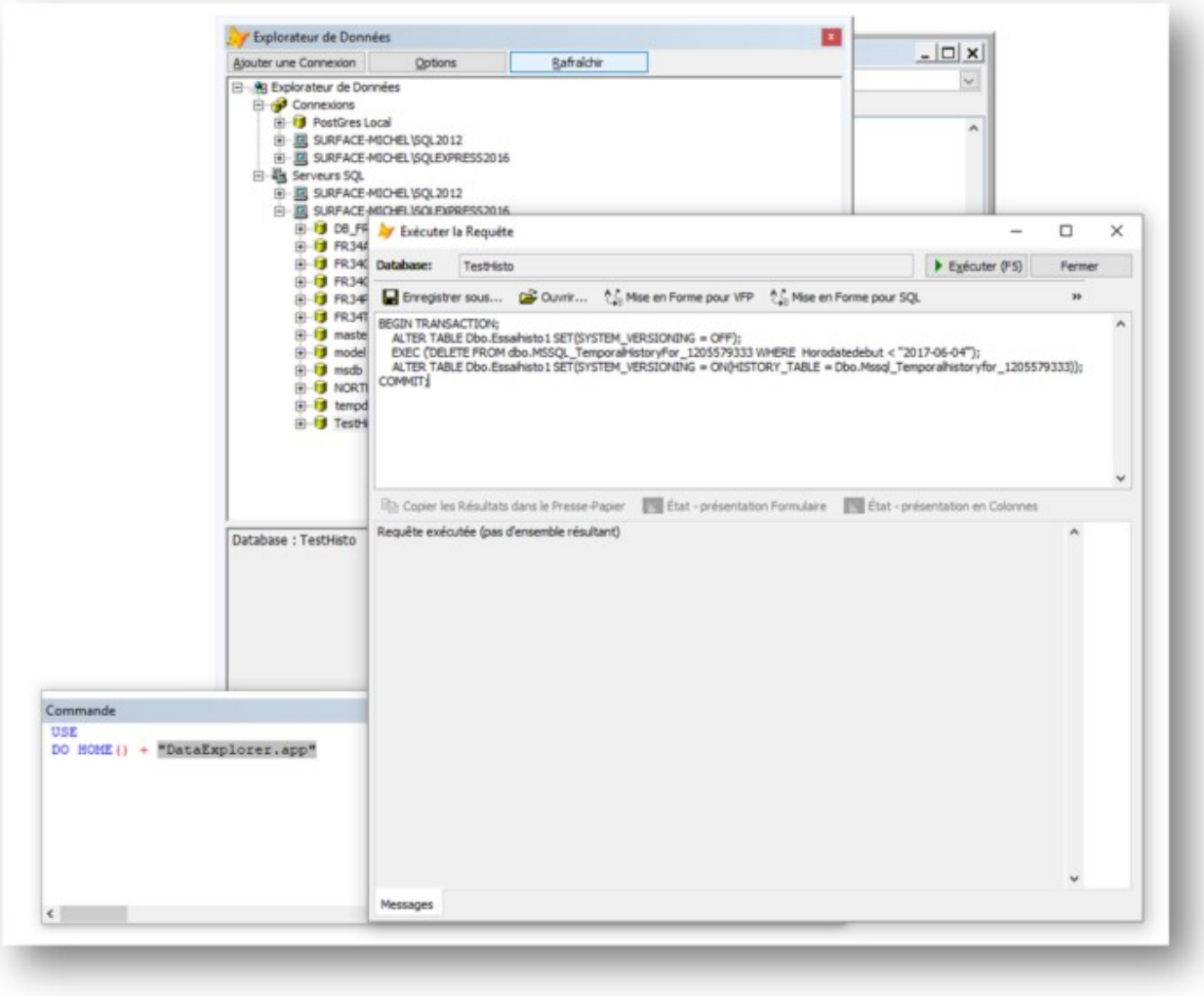
Par exemple, pour donner à la table d'historique le nom Historique_EssaiHisto1, le script de création de notre table EssaiHisto1 sera donc:

```
CREATE TABLE Essaihisto1

   (Pk_Histo1     INT IDENTITY PRIMARY KEY,
    ...,
    Horodatedebut DATETIME2 GENERATED ALWAYS AS ROW START HIDDEN NOT NULL,
    Horodatefin      DATETIME2 GENERATED ALWAYS AS ROW END     HIDDEN NOT NULL,
    PERIOD FOR SYSTEM_TIME(Horodatedebut, Horodatefin)
   )
         WITH(SYSTEM_VERSIONING = ON
                    (HISTORY_TABLE = dbo.Historique_EssaiHisto1)
               );
```

- le deuxième point à bien comprendre, est que l'instruction de DELETE de la puge doit être exécutée dans une transaction séparée. Depuis un client autre que SSMS, le seul moyen est d'inclure ce DELETE dans un ordre EXEC.
  Si nous n'avons pas nommé la table d'historique, et que nous voulons par exemple puger cet historique pour les lignes antérieures 4 jui 2017, notre script de purge sera donc être le suivant:

```
      BEGIN TRANSACTION;
          ALTER TABLE Dbo.Essaihisto1 SET(SYSTEM_VERSIONING = OFF);
          EXEC ('DELETE FROM dbo.MSSQL_TemporalHistoryFor_1205579333 WHERE Horodatedebut < ''2017-06-
04''');
          ALTER TABLE Dbo.Essaihisto1 SET(SYSTEM_VERSIONING = ON(HISTORY_TABLE =
Dbo.Mssql_Temporalhistoryfor_1205579333));
      COMMIT;
```



En conclusion

Les tables temporelles avec contrôle de version par le système sont simples à mettre en place, et ne demandent aucune modification du code existant. Il n'y a donc aucune raison de s'en priver...

Et pour en savoir plus:

[la présentation de Fred Brouard et Arian Papillon], avec des [scripts d'exemple]
en français, synthétique, complet

[l'aide en ligne de SQL Server]
pour tout savoir...

[un article de Ahmad Yaseen] (en anglais)
simple et exhasustif à la fois