```
use Transitional;
go
-------------------------------------- RELATIONAL MODEL --------------------------------------
--
--   Script by Lars Rönnbäck - latest version is always here: https://pastebin.com/AwdwCau8
--
--
--   Based on the paper "Modeling Conflicting, Unreliable, and Varying Information"
--   Full-text preprint available on ResearchGate: https://bit.ly/2A7J4Rb
--
--   This script creates a relationally modeled implementation of Transitional Modeling, with
--   some limitiations due to missing features in SQL Server. Note that this is only intended
--   as a prototype in which concepts can be tested for educational purposes.
--
--   Version: 20181220.1   Added stored procedure that simplifies data insertion.
--                         Cleaned up examples and added posit types.
--            20181217.2   Better XML in the views (old version: https://pastebin.com/kuDmLwYq).
--
--
drop procedure if exists [Assert];
drop view if exists [Check_for_Contradictions];
drop function if exists [Information_in_Effect];
drop view if exists [v_Assertion];
drop view if exists [v_Posit];
drop table if exists [Assertion];
drop table if exists [Posit];
drop table if exists [Dereference];
drop table if exists [DereferencingSet];
drop table if exists [Appearance];
drop table if exists [Role];
drop table if exists [Thing];

-- A table to store the unique identifiers of things.
-- Where a thing is that which is sufficiently distinguishable
-- from something else to be told apart.
create table [Thing] (
    [UID] uniqueidentifier not null default NEWSEQUENTIALID(),
```

```
    -- Enforce uniqueness of the unique identifiers.
    -- Note that primary keys enforce uniqueness as well as cluster
    -- the underlying table for better performance, and is needed
    -- in order to reference these using foreign key references.
    constraint [unique_and_referenceable_UID] primary key clustered (
        [UID]
    )
);

create table [Role] (
    [RoleUID] uniqueidentifier not null,
    [Role] varchar(555) not null,
    constraint [Role_is_Thing] foreign key (
        [RoleUID]
    ) references [Thing]([UID]),
    constraint [referenceable_RoleUID] primary key clustered (
        [RoleUID]
    ),
    constraint [unique_Role] unique nonclustered (
        [Role]
    )
);

/*
----------- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----------
def. of universal
A body of information is said to be universal iff positors agree on all appearances.
-------------------------------------------------------------------------------------------

In order to disagree you must first agree upon something upon which the difference in
opinion lies. At the very least, positors must agree on appearances. In other words
if two positors want to have an opinion about something, they must first come to the
conclusion that this something boils down to the same unique identifier for both of them
and that they mean the same when they talk about the roles it may appear in.

We will assume that talk about "Archie's beard" by any positor means that it is the
same Archie and the same property they are talking about.
```

```
*/
create table [Appearance] (
    [AppearanceUID] uniqueidentifier not null,
    [ThingUID] uniqueidentifier not null,
    [RoleUID] uniqueidentifier not null,
    constraint [Appearance_is_Thing] foreign key (
        [AppearanceUID]
    ) references [Thing]([UID]),
    constraint [existing_Thing] foreign key (
        [ThingUID]
    ) references [Thing]([UID]),
    constraint [existing_Role] foreign key (
        [RoleUID]
    ) references [Role]([RoleUID]),
    constraint [referenceable_AppearanceUID] primary key clustered (
        [AppearanceUID]
    ),
    constraint [unique_Appearance] unique nonclustered (
        [ThingUID],
        [RoleUID]
    )
);

create table [DereferencingSet] (
    [DereferencingSetUID] uniqueidentifier not null,
    constraint [DereferencingSet_is_Thing] foreign key (
        [DereferencingSetUID]
    ) references [Thing]([UID]),
    constraint [unique_and_referenceable_DereferencingSetUID] primary key clustered (
        [DereferencingSetUID]
    )
);

create table [Dereference] (
    [DereferencingSetUID] uniqueidentifier not null,
    [AppearanceUID] uniqueidentifier not null,
    constraint [reference_to_DereferencingSet] foreign key (
```

```
        [DereferencingSetUID]
    ) references [DereferencingSet]([DereferencingSetUID]),
    constraint [reference_to_Appearance] foreign key (
        [AppearanceUID]
    ) references [Appearance]([AppearanceUID]),
    constraint [unique_Dereference] primary key clustered (
        [DereferencingSetUID],
        [AppearanceUID]
    )
);

/*
----------- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----------
def. of canonical
A body of information is said to be canonical iff all assertions
are made against posits without negated values.
-------------------------------------------------------------------------------------------

In practice, the value of a posit may be of any data type, primitive or complex, but due to
SQL Server lacking support for generics (looks like <T> in many programming languages) we
have to limit the Value column to varchar(max). Appearance time may also be any time type,
exact or fuzzy, in order to represent that the value appeared exactly since some specified
time or inexactly within some period of time. Here the AppearanceTime column is limited
to a datetime.

Finally, in order to represent information, canonical form is used, which simply means that
values are stored without negation, for example "red" is acceptable, and "not red" is not.
Opposite opinions are instead handled using negative Reliability in assertions.
*/
create table [Posit] (
    [PositUID] uniqueidentifier not null,
    [DereferencingSetUID] uniqueidentifier not null,
    [Value] varchar(max) null,
    [AppearanceTime] datetime not null,
    constraint [existing_DereferencingSet] foreign key (
        [DereferencingSetUID]
    ) references [DereferencingSet]([DereferencingSetUID]),
```

```
    constraint [Posit_is_Thing] foreign key (
        [PositUID]
    ) references [Thing]([UID]),
    constraint [referenceable_PositUID] primary key clustered (
        [PositUID]
    )
);

/*
----------- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----------
def. of a body of information
A body of information is a set of true assertions.

def. of exclusive
A body of information is said to be exclusive iff no assertions
in which only the reliability differs exist.
-------------------------------------------------------------------------------------------

We will assume that any data inserted into this table represents true assertions. We will not
constrain the Reliability column, but in practice it should be in the interval [-1, 1].
The constraint [unique_Assertion] ensures that the stored assertions are exclusive, so that
there is no ambiguity with respect to the assigned Reliability.
*/
create table [Assertion] (
    [AssertionUID] uniqueidentifier not null,
    [PositorUID] uniqueidentifier not null,
    [PositUID] uniqueidentifier not null,
    [Reliability] decimal(3,2) not null,
    [AssertionTime] datetime not null,
    constraint [Assertion_is_Thing] foreign key (
        [AssertionUID]
    ) references [Thing]([UID]),
    constraint [existing_Positor] foreign key (
        [PositorUID]
    ) references [Thing]([UID]),
    constraint [existing_Posit] foreign key (
        [PositUID]
```

```
    ) references [Posit]([PositUID]),
    constraint [referenceable_AssertionUID] primary key nonclustered (
        [AssertionUID]
    ),
    constraint [unique_Assertion] unique clustered (
        [PositorUID],
        [PositUID],
        [AssertionTime]
    )
);

go
create or alter view [v_Posit]
as
select
    p.[PositUID],
    p.[DereferencingSetUID],
    p.[Value],
    p.[AppearanceTime],
    (
        select
            p.[PositUID] as [@PositUID],
            p.[AppearanceTime] as [@AppearanceTime],
            p.[DereferencingSetUID] as [DereferencingSet/@DereferencingSetUID],
            (
                select
                    a.[AppearanceUID] as [@AppearanceUID],
                    a.[ThingUID] as [@ThingUID],
                    r.[Role] as [@Role]
                from
                    [DereferencingSet] s
                join
                    [Dereference] d
                on
                    d.[DereferencingSetUID] = s.[DereferencingSetUID]
                join
                    [Appearance] a
```

```
                on
                    a.[AppearanceUID] = d.[AppearanceUID]
                join
                    [Role] r
                on
                    r.[RoleUID] = a.[RoleUID]
                where
                    s.[DereferencingSetUID] = p.[DereferencingSetUID]
                order by
                    r.[RoleUID] asc
                for xml path('Appearance'), type
            ) as [DereferencingSet],
            p.[Value]
        for xml path('Posit'), type
    ) as [PositXML]
from
    [Posit] p

go
create or alter view [v_Assertion]
as
select
    a.[AssertionUID],
    a.[PositorUID],
    a.[PositUID],
    p.[DereferencingSetUID],
    p.[Value],
    p.[AppearanceTime],
    a.[Reliability],
    a.[AssertionTime],
    (
        select
            a.[AssertionUID] as [@AssertionUID],
            a.[PositorUID] as [@PositorUID],
            a.[Reliability] as [@Reliability],
            a.[AssertionTime] as [@AssertionTime],
            p.[PositXML].query('.')
```

```
        for xml path('Assertion'), type
    ) as [AssertionXML]
from
    [Assertion] a
join
    [v_Posit] p
on
    p.[PositUID] = a.[PositUID]
go

/*
----------------------------------------- XML FORMAT -----------------------------------------
  Specify UID for already known things and some Label for unknown. Bracketed attributes
  are optional. Labels will result in new things being created, one for each unique label.

    <Assertion PositorUID="<UID|Label>" Reliability="1.00" AssertionTime="2018-12-13T15:30:00">
      <Posit [PositUID="<UID>"] AppearanceTime="2018-12-01T12:00:00">
        <DereferencingSet [DereferencingSetUID="<UID>"]>
          <Appearance [AppearanceUID="<UID>"] ThingUID="<UID|Label>" Role="has beard" />
        </DereferencingSet>
        <Value>fluffy red</Value>
      </Posit>
    </Assertion>

  Note that AssertionUID cannot be specified. Assertions can never be altered once they've been
  created.
-----------------------------------------------------------------------------------------------
*/
create or alter procedure Assert (
    @assertion xml OUTPUT
)
as
begin
    set nocount on;

    -- create a lookup table
    create table #Thing (
```

```
        [ThingUID] uniqueidentifier not null primary key,
        [UID_or_Label] varchar(555) not null
    );

    -- is a posit set already referenced through a UID?
    declare @PositUID uniqueidentifier = @assertion.value(
        '/Assertion[1]/Posit[1]/@PositUID',
        'uniqueidentifier'
    );

    -- if not it needs to be built
    if @PositUID is null
    begin
        -- is a dereferencing set already referenced through a UID?
        declare @DereferencingSetUID uniqueidentifier = @assertion.value(
            '/Assertion[1]/Posit[1]/DereferencingSet[1]/@DereferencingSetUID',
            'uniqueidentifier'
        );

        if @DereferencingSetUID is not null
        begin
            if not exists (
                select [DereferencingSetUID] from [DereferencingSet]
                where [DereferencingSetUID] = @DereferencingSetUID
            )
            raiserror('The referenced @DereferencingSetUID does not exist', 10, 1);
        end
        else
        begin
            -- check if given AppareanceUID:s exist
            if exists (
                select x.[AppearanceUID]
                from (
                    select distinct
                        Appearance.value('@AppearanceUID', 'uniqueidentifier') as [AppearanceUID]
                    from @assertion.nodes(
                        '/Assertion/Posit/DereferencingSet/Appearance[@AppearanceUID]'
```

```
                ) as T(Appearance)
            ) x
            left join [Appearance] a
            on a.[AppearanceUID] = x.[AppearanceUID]
            where a.[AppearanceUID] is null
        )
        raiserror('The referenced @AppearanceUID does not exist', 10, 1);

        -- create a table to hold appearances
        create table #Appearance (
            AppearanceUID uniqueidentifier not null primary key,
            ThingUID uniqueidentifier not null,
            RoleUID uniqueidentifier not null
        );

        -- add existing and referenced appearances
        insert into #Appearance
        select a.*
        from (
            select distinct
                Appearance.value('@AppearanceUID', 'uniqueidentifier') as [AppearanceUID]
            from @assertion.nodes(
                '/Assertion/Posit/DereferencingSet/Appearance[@AppearanceUID]'
            ) as T(Appearance)
        ) x
        join [Appearance] a
        on a.[AppearanceUID] = x.[AppearanceUID];

        declare @matches int = 0;
        -- create missing things and populate the lookup table
        merge [Thing] t
        using (
            select distinct
                Appearance.value('@ThingUID', 'varchar(555)') as [UID_or_Label]
            from @assertion.nodes(
                '/Assertion/Posit/DereferencingSet/Appearance[not(@AppearanceUID)]'
            ) as T(Appearance)
```

```
        ) i on i.[UID_or_Label] = cast(t.[UID] as varchar(555))
        when not matched then insert values(DEFAULT)
        when matched then update set @matches = @matches + 1
        output inserted.[UID], i.[UID_or_Label] into #Thing;

        -- create a lookup table
        create table #Role (
            [RoleUID] uniqueidentifier not null primary key,
            [Role] varchar(555) not null
        );

        -- lookup or create the roles
        merge [Thing] t
        using (
            select
                r.[RoleUID],
                x.[Role]
            from (
                select
                    Appearance.value('@Role', 'varchar(555)') as [Role]
                from @assertion.nodes(
                    '/Assertion/Posit/DereferencingSet/Appearance[not(@AppearanceUID)]'
                ) as T(Appearance)
            ) x
            left join [Role] r
            on x.[Role] = r.[Role]
        ) i on i.[RoleUID] = t.[UID]
        when not matched then insert values(DEFAULT)
        when matched then update set @matches = @matches + 1
        output inserted.[UID], i.[Role] into #Role;

        -- add missing roles to Role
        merge [Role] r
        using #Role rlkp
        on rlkp.[RoleUID] = r.[RoleUID]
        when not matched then insert values (rlkp.[RoleUID], rlkp.[Role]);

```

```
            -- lookup or create the appearances
            merge [Thing] t
            using (
                select
                    a.[AppearanceUID],
                    tlkp.[ThingUID],
                    rlkp.[RoleUID]
                from (
                    select
                        Appearance.value('@ThingUID', 'varchar(555)') as [UID_or_Label],
                        Appearance.value('@Role', 'varchar(555)') as [Role]
                    from @assertion.nodes(
                        '/Assertion/Posit/DereferencingSet/Appearance[not(@AppearanceUID)]'
                    ) as T(Appearance)
                ) x
                join #Thing tlkp
                on tlkp.[UID_or_Label] = x.[UID_or_Label]
                join #Role rlkp
                on rlkp.[Role] = x.[Role]
                left join [Appearance] a
                on a.[ThingUID] = tlkp.[ThingUID] and a.[RoleUID] = rlkp.[RoleUID]
            ) i on i.[AppearanceUID] = t.[UID]
            when not matched then insert values(DEFAULT)
            when matched then update set @matches = @matches + 1
            output inserted.[UID], i.[ThingUID], i.[RoleUID] into #Appearance;

            -- add missing apperances to Appearance
            merge [Appearance] a
            using #Appearance alkp
            on alkp.[AppearanceUID] = a.[AppearanceUID]
            when not matched then insert values (alkp.[AppearanceUID], alkp.[ThingUID], alkp.[RoleUID]);

            declare @NumberOfAppearances int = (select count(*) from #Appearance);

            -- do these appearances already exist as a dereferencing set?
            set @DereferencingSetUID = (
                select d.DereferencingSetUID
```

```
                  from [Dereference] d
                left join #Appearance a on a.AppearanceUID = d.AppearanceUID
                group by d.DereferencingSetUID
                having COUNT(d.AppearanceUID) = @NumberOfAppearances
                   and COUNT(a.AppearanceUID) = @NumberOfAppearances
            );

            -- if not it needs to be created
            if @DereferencingSetUID is null
            begin
                create table #DereferencingSet (
                    [DereferencingSetUID] uniqueidentifier not null primary key
                );

                insert into [Thing]
                output inserted.[UID] into #DereferencingSet
                values (DEFAULT);

                set @DereferencingSetUID = (select top 1 [DereferencingSetUID] from #DereferencingSet);

                insert into [DereferencingSet] values (@DereferencingSetUID);

                insert into [Dereference]
                select @DereferencingSetUID, [AppearanceUID]
                from #Appearance;
            end
        end -- end of building a dereference set
        -- now we have a @DereferencingSetUID

        declare @Value varchar(max);
        declare @AppearanceTime datetime;

        select
            @Value = cast(Posit.query('Value/node()') as varchar(max)),
            @AppearanceTime = Posit.value('@AppearanceTime', 'datetime')
        from @assertion.nodes(
            '/Assertion[1]/Posit[1]'
```

```
        ) as T(Posit);

        -- check if it already exists
        set @PositUID = (
            select [PositUID]
            from [Posit]
            where [DereferencingSetUID] = @DereferencingSetUID
              and [Value] = @Value
              and [AppearanceTime] = @AppearanceTime
        );

        if @PositUID is null
        begin
            -- create the posit
            create table #Posit (
                [PositUID] uniqueidentifier not null primary key
            );

            insert into [Thing]
            output inserted.[UID] into #Posit
            values (DEFAULT);

            set @PositUID = (select top 1 [PositUID] from #Posit);

            insert into [Posit] (
                [PositUID],
                [DereferencingSetUID],
                [Value],
                [AppearanceTime]
            )
            values (
                @PositUID,
                @DereferencingSetUID,
                @Value,
                @AppearanceTime
            );
        end -- new posit created
```

```
    end -- end of building posit
    -- now we have a @PositUID

    declare @AssertionUID uniqueidentifier;

    -- create the assertion
    create table #Assertion (
        [AssertionUID] uniqueidentifier not null primary key
    );

    insert into [Thing]
    output inserted.[UID] into #Assertion
    values (DEFAULT);

    set @AssertionUID = (select top 1 [AssertionUID] from #Assertion);

    -- A positor must be referenced through a UID or Label
    declare @PositorUID_or_Label varchar(555) = @assertion.value(
        '/Assertion[1]/@PositorUID',
        'varchar(555)'
    );

    if @PositorUID_or_Label is null
    raiserror('No @PositorUID has been specified', 10, 1);

    -- is must either be a UID or a Label (and if it is a Label it is already created)
    declare @PositorUID uniqueidentifier = isnull(
        (select [ThingUID] from #Thing where [UID_or_Label] = @PositorUID_or_Label),
        (select [UID] from [Thing] where cast([UID] as varchar(555)) = @PositorUID_or_Label)
    );

    if @PositorUID is null
    raiserror('The referenced @PositorUID does not exist', 10, 1);

    -- now store the assertion
    insert into [Assertion] (
        [AssertionUID],
```

```
        [PositorUID],
        [PositUID],
        [Reliability],
        [AssertionTime]
    )
    select
        @AssertionUID,
        @PositorUID,
        @PositUID,
        Posit.value('@Reliability', 'decimal(3,2)') as [AppearanceTime],
        Posit.value('@AssertionTime', 'datetime') as [AppearanceTime]
    from @assertion.nodes(
        '/Assertion[1]'
    ) as T(Posit)

    -- and we are done, so finally set the output variable
    set @assertion = (
        select [AssertionXML] from [v_Assertion] where [AssertionUID] = @AssertionUID
    );
end
go

-------------------------------------- PROPERTIES --------------------------------------
-- first create a few things (and positors), and relate them to a name using "has name"
declare @assertion1 xml = '
    <Assertion PositorUID="A" Reliability="1.00" AssertionTime="2018-12-15 00:00">
      <Posit AppearanceTime="1972-08-20 15:30">
        <DereferencingSet>
          <Appearance ThingUID="A" Role="has name" />
        </DereferencingSet>
        <Value>Archie</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion1 OUTPUT;

-- the stored assertion is returned (in case we need any of the UID:s for later)
```

```
select @assertion1;
-- but we can also look at it using the view
select * from [v_Assertion];

declare @assertion2 xml = '
    <Assertion PositorUID="B" Reliability="1.00" AssertionTime="2018-12-15 00:00">
      <Posit AppearanceTime="1972-02-13 08:00">
        <DereferencingSet>
          <Appearance ThingUID="B" Role="has name" />
        </DereferencingSet>
        <Value>Bella</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion2 OUTPUT;

declare @assertion3 xml = '
    <Assertion PositorUID="M" Reliability="1.00" AssertionTime="2018-12-15 00:00">
      <Posit AppearanceTime="1945-10-17 12:00">
        <DereferencingSet>
          <Appearance ThingUID="M" Role="has name" />
        </DereferencingSet>
        <Value>Modeler</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion3 OUTPUT;

declare @assertion4 xml = '
    <Assertion PositorUID="S" Reliability="1.00" AssertionTime="2018-12-15 00:00">
      <Posit AppearanceTime="2018-12-15">
        <DereferencingSet>
          <Appearance ThingUID="S" Role="has name" />
        </DereferencingSet>
        <Value>Script</Value>
      </Posit>
    </Assertion>
```

```
';
exec Assert @assertion4 OUTPUT;

declare @assertion5 xml = '
    <Assertion PositorUID="D" Reliability="1.00" AssertionTime="2018-12-15 00:00">
      <Posit AppearanceTime="2001-01-01 01:00">
        <DereferencingSet>
          <Appearance ThingUID="D" Role="has name" />
        </DereferencingSet>
        <Value>Disser</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion5 OUTPUT;

-- there should now be five assertions
select * from [v_Assertion];

-- first we will let Bella assert a few things about Archie
declare @ArchieUID6 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID6 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion6 xml = '
    <Assertion PositorUID="' + @BellaUID6 + '" Reliability="1.00" AssertionTime="2018-12-13 15:30">
      <Posit AppearanceTime="2018-12-01 12:00">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID6 + '" Role="has beard" />
        </DereferencingSet>
        <Value>fluffy red</Value>
      </Posit>
    </Assertion>
';
```

```
exec Assert @assertion6 OUTPUT;

-- there should now be a new assertion
select * from [v_Assertion] where [Value] = 'fluffy red';

-- let the Disser have a different opinion
declare @ArchieUID7 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @DisserUID7 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion7 xml = '
    <Assertion PositorUID="' + @DisserUID7 + '" Reliability="1.00" AssertionTime="2018-12-13 15:30">
      <Posit AppearanceTime="2018-12-01 12:00">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID7 + '" Role="has beard" />
        </DereferencingSet>
        <Value>shaved clean</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion7 OUTPUT;

-- there should now be two assertions
-- where Bella and the Disagreer have conflicting views of Archie's beard
select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean');

-- let the Disser further oppose to what Bella stated
declare @ArchieUID8 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @DisserUID8 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
```

```
    cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion8 xml = '
    <Assertion PositorUID="' + @DisserUID8 + '" Reliability="-1.00" AssertionTime="2018-12-13 15:35">
      <Posit AppearanceTime="2018-12-01 12:00">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID8 + '" Role="has beard" />
        </DereferencingSet>
        <Value>fluffy red</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion8 OUTPUT;

-- so Bella and the Disser have opposing views of the posit stating it is fluffy red
select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean');

-- let Bella assert that there is a very small chance it actually was shaved clean
declare @ArchieUID9 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID9 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion9 xml = '
    <Assertion PositorUID="' + @BellaUID9 + '" Reliability="0.05" AssertionTime="2018-12-13 15:35">
      <Posit AppearanceTime="2018-12-01 12:00">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID9 + '" Role="has beard" />
        </DereferencingSet>
        <Value>shaved clean</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion9 OUTPUT;
```

```
741.
742. -- Bella asserts that it is very unlikely, but Archie's beard may have been shaved clean
743. select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean')
744. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
745.
746. -- she is sure Archie was shaved clean later though (a real change occured between 12 and 13)
747. declare @ArchieUID10 varchar(555) = (
748.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
749.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
750. );
751. declare @BellaUID10 varchar(555) = (
752.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
753.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
754. );
755. declare @assertion10 xml = '
756.     <Assertion PositorUID="' + @BellaUID10 + '" Reliability="1.00" AssertionTime="2018-12-13 15:35">
757.       <Posit AppearanceTime="2018-12-01 13:00">
758.         <DereferencingSet>
759.           <Appearance ThingUID="' + @ArchieUID10 + '" Role="has beard" />
760.         </DereferencingSet>
761.         <Value>shaved clean</Value>
762.       </Posit>
763.     </Assertion>
764. ';
765. exec Assert @assertion10 OUTPUT;
766.
767. -- Bella asserts that at 13:00 Archie had shaved
768. select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean')
769. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
770.
771. /*
772. ----------- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----------
773. def. of the information in effect
774. Let A be a body of information. The information in effect is a
775. subset A(T@, t@) ⊂ A given a bitemporal point in assertion
776. and appearance time (T@, t@) ∈ T × t. Assuming P, α, T, D, v,
777. and t are free variables, all assertions !(P, p, α, T) ∈ A(T@, t@)
```

```
with p = [D, v, t] are found using the following selection
criteria:
1. Let A1 ⊂ A be the assertions in A satisfying T ≤ T@ and
   t ≤ t@.
2. Let A2 ⊂ A1 be the assertions in A1 with α 6= 0 and the
   latest appearance time t for each combination of P and
   D, excluding those assertions !(P, p, α, T) for which an
   assertion !(P, p, 0, T0) exists with T ≤ T0 ≤ T@.
3. Let A(T@, t@) ⊂ A2 be the assertions from A2 with the
   latest assertion time T for each combination of P, D, and v.
-------------------------------------------------------------------------------------------

A function for the information in effect is created below according
to the selection criteria defined above.
*/
go
create or alter function [Information_in_Effect] (
    @appearanceTime datetime,
    @assertionTime datetime
)
returns table as return
with A1 as (
    select *
      from [v_Assertion]
     where [AppearanceTime] <= @appearanceTime
       and [AssertionTime]  <= @assertionTime
),
A2 as (
    select a.*
      from A1 a
     where a.[Reliability] <> 0
       and a.[AppearanceTime] = (
           select max(s.[AppearanceTime])
             from A1 s
            where s.[PositorUID] = a.[PositorUID]
              and s.[DereferencingSetUID] = a.[DereferencingSetUID]
       )
```

```
       and not exists (
           select x.[AssertionUID]
             from A1 x
            where x.[PositorUID] = a.[PositorUID]
              and x.[PositUID] = a.[PositUID]
              and x.[Reliability] = 0
              and x.[AssertionTime] between a.AssertionTime and @assertionTime
       )
)
select a.*
  from A2 a
 where a.[AssertionTime] = (
       select max(s.[AssertionTime])
         from A2 s
        where s.[PositorUID] = a.[PositorUID]
          and s.[DereferencingSetUID] = a.[DereferencingSetUID]
          and s.[Value] = a.[Value]
)
go

-- which information is in effect at 12 and 13 given what was asserted on or before 15:30?
select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:30')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:30')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- which information is in effect at 12 and 13 given what was asserted on or before 15:35?
select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:35')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:35')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

/*
----------- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----------
def. of symmetric
A body of information is said to be symmetric iff the assertions
!(P, p, -α, T) and !(P, p̄, α, T) are equivalent for a posit p and its opposite p̄.
```

```
def. of bounded
A body of information is said to be bounded iff the reliabilities in the assertions
!(P, p, α, T) and !(P, p̄, β, T) satisfy |α + β| = |α + β|^2.

def. of non-contradictory
A body of information is said to be non-contradictory iff for
every information in effect the reliabilities, numbered 1 to n,
in all positive and negative assertions made by the same positor
for the same dereferencing set satisfy the inequality:
    n                  n
1/2 ∑ (1 - sign(αi)) + ∑ αi ≤ 1
   i=1                i=1
-----------------------------------------------------------------------------------------

The fact that we can talk about contradictions relies on the assumption that our body
of information is symmetric. When a body of information is also bounded it is possible to
transform all information into canonical form (negated values like "not red" are instead
expressed using "red" and a negative Reliability).
*/

-- we create a view that checks for contradictions according to the inequality above
go
create or alter view [Check_for_Contradictions]
as
with bitemporalTimepoints as (
    select [AppearanceTime], [AssertionTime]
    from (select distinct [AppearanceTime] from Posit) p
    cross join (select distinct [AssertionTime] from Assertion) a
)
select
    ineq.*,
    case
        when InequalitySum <= 1 then 'Non-Contradictory'
        else 'Contradictory'
    end as Result
from (
```

```
    select
        t.[AppearanceTime],
        t.[AssertionTime],
        iie.[PositorUID],
        iie.[DereferencingSetUID],
        0.5 * sum(1 - sign(iie.[Reliability])) + sum(iie.[Reliability]) as InequalitySum
    from
        bitemporalTimepoints t
    cross apply
        [Information_in_Effect](t.[AppearanceTime], t.[AssertionTime]) iie
    group by
        t.[AppearanceTime],
        t.[AssertionTime],
        iie.[PositorUID],
        iie.[DereferencingSetUID]
) ineq;
go

-- have contradictory assertions been made?
select * from [Check_for_Contradictions];
-- Bella is contradictory for the appearance time between 12 and 13 for assertion time 15:35 and onwards
-- so with our current knowledge she has made contradictory statements

-- Bella realizes she will remain forever contradictory unless she makes another assertion
-- Bella revaluates her earlier assertion from 1 to 0.95
declare @ArchieUID11 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID11 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion11 xml = '
    <Assertion PositorUID="' + @BellaUID11 + '" Reliability="0.95" AssertionTime="2018-12-13 15:40">
      <Posit AppearanceTime="2018-12-01 12:00">
        <DereferencingSet>
```

```
          <Appearance ThingUID="' + @ArchieUID11 + '" Role="has beard" />
        </DereferencingSet>
        <Value>fluffy red</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion11 OUTPUT;

-- which information is in effect at 12 and 13 given that assertion
select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:40')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:40')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
-- now the situation looks better for Bella between 12 and 13

-- have contradictory assertions been made?
select * from [Check_for_Contradictions];
-- Bella is contradictory for the appearance time between 12 and 13 but now only for
-- assertion time between 15:35 and 15:45

-- the Disagreer is now changing it's mind and retracts the previous statement
-- a retraction has reliability = 0
declare @ArchieUID12 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @DisserUID12 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion12 xml = '
    <Assertion PositorUID="' + @DisserUID12 + '" Reliability="0" AssertionTime="2018-12-13 15:40">
      <Posit AppearanceTime="2018-12-01 12:00">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID12 + '" Role="has beard" />
        </DereferencingSet>
        <Value>shaved clean</Value>
```

```
      </Posit>
    </Assertion>
';
exec Assert @assertion12 OUTPUT;

-- which information is now in effect at 12 and 13 at the time of the retraction?
select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:40')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:40')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
-- the Disser is now only saying it's not fluffy red

-- Bella is going to reiterate that Archie's beard was also shaved clean one hour later
declare @ArchieUID13 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID13 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion13 xml = '
    <Assertion PositorUID="' + @BellaUID13 + '" Reliability="1.00" AssertionTime="2018-12-13 15:45">
      <Posit AppearanceTime="2018-12-01 14:00">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID13 + '" Role="has beard" />
        </DereferencingSet>
        <Value>shaved clean</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion13 OUTPUT;

-- the latest assertion is called a "restatement", since the value is not changing
select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

```

```
-- the information in effect will show the latest restatement
select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:45')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:45')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
select * from [Information_in_Effect]('2018-12-01 14:00', '2018-12-13 15:45')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- Bella is going to reassure that she still is certain Archie's beard was shaved clean at 14
declare @ArchieUID14 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID14 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion14 xml = '
    <Assertion PositorUID="' + @BellaUID14 + '" Reliability="1.00" AssertionTime="2018-12-13 15:50">
      <Posit AppearanceTime="2018-12-01 14:00">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID14 + '" Role="has beard" />
        </DereferencingSet>
        <Value>shaved clean</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion14 OUTPUT;

-- the latest assertion is called a "reassertion", since only the assertion time changes
select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- the information in effect will show the latest restatement and latest reassertion
select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:45')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:45')
```

```
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
select * from [Information_in_Effect]('2018-12-01 14:00', '2018-12-13 15:45')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
select * from [Information_in_Effect]('2018-12-01 14:00', '2018-12-13 15:50')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-------------------------------------- METADATA --------------------------------------
-- The good thing about posits, assertions, and so on being "things" in themselves
-- is that it makes it possible to produce posits about them as well. This is
-- usually called metadata.

-- let the Script positor assert when what we have done so far was recorded in
-- our database right now.

-- we now have 42 things that we want to create metadata for
select * from [Thing];

declare @ScriptUID15 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Script"]//*[@Role = "has name"]') as T(Appearance)
);

declare thing cursor static for (select [UID] from [Thing]);
open thing;

declare @ThingUID15 varchar(555);
fetch next from thing into @ThingUID15;

declare @now char(19) = CONVERT(varchar(19), getdate(), 121);

while(@@FETCH_STATUS = 0)
begin
    declare @assertion15 xml = '
        <Assertion PositorUID="' + @ScriptUID15 + '" Reliability="1.00" AssertionTime="' + @now + '">
          <Posit AppearanceTime="' + @now + '">
            <DereferencingSet>
              <Appearance ThingUID="' + @ThingUID15 + '" Role="was recorded at" />
```

```
            </DereferencingSet>
            <Value>' + @now + '</Value>
          </Posit>
        </Assertion>
    ';
    exec Assert @assertion15 OUTPUT;

    fetch next from thing into @ThingUID15;
end

close thing;
deallocate thing;

-- now there are 42 assertions from which we can tell when the data entered the database
select * from [v_Assertion] where isdate([Value]) = 1
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- note that the metadata is not in effect if we travel back in time
select * from [Information_in_Effect]('2018-12-01 14:00', '2018-12-13 15:50')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- but it is in effect now
select * from [Information_in_Effect](getdate(), getdate())
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-------------------------------------- RELATIONSHIPS --------------------------------------
-- we will start by marrying Archie and Bella in the Notre-Dame
-- so we first need the Notre-Dame thing
declare @assertion16 xml = '
    <Assertion PositorUID="NDame" Reliability="1.00" AssertionTime="2018-12-15 00:00">
      <Posit AppearanceTime="1753-01-01">
        <DereferencingSet>
          <Appearance ThingUID="NDame" Role="has name" />
        </DereferencingSet>
        <Value>Notre-Dame de Paris</Value>
      </Posit>
    </Assertion>
```

```
';
exec Assert @assertion16 OUTPUT;

-- we need the UID:s
declare @ArchieUID17 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID17 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
declare @NDameUID17 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Notre-Dame de Paris"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion17 xml = '
    <Assertion PositorUID="' + @BellaUID17 + '" Reliability="1.00" AssertionTime="2018-12-13 15:55">
      <Posit AppearanceTime="2004-06-19 15:00">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID17 + '" Role="husband" />
          <Appearance ThingUID="' + @BellaUID17 + '" Role="wife" />
          <Appearance ThingUID="' + @NDameUID17 + '" Role="church" />
        </DereferencingSet>
        <Value>married</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion17 OUTPUT;

-- now Bella has asserted the marriage
select * from [v_Assertion] where [Value] in ('married')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- let them divorce a while later (using the same dereferencing set, so it is a real change)
declare @ArchieUID18 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
```

```
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID18 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
declare @NDameUID18 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Notre-Dame de Paris"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion18 xml = '
    <Assertion PositorUID="' + @BellaUID18 + '" Reliability="1.00" AssertionTime="2018-12-13 15:55">
      <Posit AppearanceTime="2010-01-31 10:00">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID18 + '" Role="husband" />
          <Appearance ThingUID="' + @BellaUID18 + '" Role="wife" />
          <Appearance ThingUID="' + @NDameUID18 + '" Role="church" />
        </DereferencingSet>
        <Value>divorced</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion18 OUTPUT;

-- now Bella has asserted the divorce as well
select * from [v_Assertion] where [Value] in ('married', 'divorced')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- however, as it turns out, Bella remarried the Disagreer at a later time, but not in a church
-- that means a new dereferencing set is needed and this time without the church
declare @DisserUID19 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID19 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
```

```
);
declare @assertion19 xml = '
    <Assertion PositorUID="' + @BellaUID19 + '" Reliability="1.00" AssertionTime="2018-12-13 15:55">
      <Posit AppearanceTime="2011-11-11 11:11">
        <DereferencingSet>
          <Appearance ThingUID="' + @DisserUID19 + '" Role="husband" />
          <Appearance ThingUID="' + @BellaUID19 + '" Role="wife" />
        </DereferencingSet>
        <Value>married</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion19 OUTPUT;

-- now Bella has asserted her second marriage, but her second marriage took place without a church
select * from [v_Assertion] where [Value] in ('married', 'divorced')
order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- another way to find this is from finding every assertion in which Bella has appeared in the wife role
-- regardless if the relationship involves a church or not
declare @BellaUID_Q1 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
select * from [v_Assertion]
where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q1") and @Role = "wife"]') = 1
order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
-- the above can be achieved through joins in the underlying model as well

-- but, Bella made a mistake, the appearance time of her second marriage is not correct
-- so she first makes a retraction of the erroneous assertion
declare @DisserUID20 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID20 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
```

```
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion20 xml = '
    <Assertion PositorUID="' + @BellaUID20 + '" Reliability="0" AssertionTime="2018-12-13 16:00">
      <Posit AppearanceTime="2011-11-11 11:11">
        <DereferencingSet>
          <Appearance ThingUID="' + @DisserUID20 + '" Role="husband" />
          <Appearance ThingUID="' + @BellaUID20 + '" Role="wife" />
        </DereferencingSet>
        <Value>married</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion20 OUTPUT;

-- in the view of all assertions the retraction is now visible
declare @BellaUID_Q2 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
select * from [v_Assertion]
where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q2") and @Role = "wife"]') = 1
order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- which means that the latest information we have is that Bella is divorced
select * from [Information_in_Effect](getdate(), getdate())
where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q2") and @Role = "wife"]') = 1
order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- so Bella needs to assert a different posit, with the correct appearance time
declare @DisserUID21 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID21 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
```

```
);
declare @assertion21 xml = '
    <Assertion PositorUID="' + @BellaUID21 + '" Reliability="1.00" AssertionTime="2018-12-13 16:00">
      <Posit AppearanceTime="2012-12-12 12:12">
        <DereferencingSet>
          <Appearance ThingUID="' + @DisserUID21 + '" Role="husband" />
          <Appearance ThingUID="' + @BellaUID21 + '" Role="wife" />
        </DereferencingSet>
        <Value>married</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion21 OUTPUT;

-- in the view of all assertions the correction is now visible
declare @BellaUID_Q3 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
select * from [v_Assertion]
where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q3") and @Role = "wife"]') = 1
order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- new the latest information we have is that Bella is married, but note that the earlier divorce is also shown
-- due to it having a different dereferencing set
select * from [Information_in_Effect](getdate(), getdate())
where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q3") and @Role = "wife"]') = 1
order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
-- there is currently no way for the system to know that these two relationships (with and without a church) both
-- represent a marriage


------------------------------------------ MODELING ----------------------------------------
/*
----------- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----------
def. of a classifier and a class
Let "is class" be a role reserved for the purpose of modeling.
```

```
A posit pc = [{(C, is class)}, c, t], defines the name
of a class through the string c and associates the unique
identifier C with it. A classifier is a relationship that binds
a thing to a class, expressed through posits on the form
pM = [{(i, thing),(C, class)}, v, t].
-----------------------------------------------------------------------------------------

It is time for the Modeler to step in and tell us what some of our things are.
*/

-- first create three classes: Person, Cathedral, and Church
declare @ModelerUID22 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
);
declare @assertion22A xml = '
    <Assertion PositorUID="' + @ModelerUID22 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
          <Appearance ThingUID="C" Role="is class" />
        </DereferencingSet>
        <Value>Person</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion22A OUTPUT;
declare @assertion22B xml = '
    <Assertion PositorUID="' + @ModelerUID22 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
          <Appearance ThingUID="C" Role="is class" />
        </DereferencingSet>
        <Value>Cathedral</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion22B OUTPUT;
```

```
declare @assertion22C xml = '
    <Assertion PositorUID="' + @ModelerUID22 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
          <Appearance ThingUID="C" Role="is class" />
        </DereferencingSet>
        <Value>Church</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion22C OUTPUT;

-- list all classes
select a.* from [v_Assertion] a
where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') = 1
order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

-- now associate some things with the classes
declare @ModelerUID23 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
);
declare @ArchieUID23 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
);
declare @BellaUID23 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
);
declare @NDameUID23 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Notre-Dame de Paris"]//*[@Role = "has name"]') as T(Appearance)
);
declare @PersonUID23 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Person"]//*[@Role = "is class"]') as T(Appearance)
```

```
);
declare @CathedralUID23 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Cathedral"]//*[@Role = "is class"]') as T(Appearance)
);
declare @ChurchUID23 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Church"]//*[@Role = "is class"]') as T(Appearance)
);
declare @assertion23A xml = '
    <Assertion PositorUID="' + @ModelerUID23 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
          <Appearance ThingUID="' + @ArchieUID23 + '" Role="thing" />
          <Appearance ThingUID="' + @PersonUID23 + '" Role="class" />
        </DereferencingSet>
        <Value>active</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion23A OUTPUT;
declare @assertion23B xml = '
    <Assertion PositorUID="' + @ModelerUID23 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
          <Appearance ThingUID="' + @BellaUID23 + '" Role="thing" />
          <Appearance ThingUID="' + @PersonUID23 + '" Role="class" />
        </DereferencingSet>
        <Value>active</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion23B OUTPUT;
declare @assertion23C xml = '
    <Assertion PositorUID="' + @ModelerUID23 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
```

```
          <Appearance ThingUID="' + @NDameUID23 + '" Role="thing" />
          <Appearance ThingUID="' + @CathedralUID23 + '" Role="class" />
        </DereferencingSet>
        <Value>active</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion23C OUTPUT;
declare @assertion23D xml = '
    <Assertion PositorUID="' + @ModelerUID23 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
          <Appearance ThingUID="' + @NDameUID23 + '" Role="thing" />
          <Appearance ThingUID="' + @ChurchUID23 + '" Role="class" />
        </DereferencingSet>
        <Value>active</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion23D OUTPUT;

-- list all classes and classifiers
-- as can be seen there are two things of class Person and one thing of class Cathedral and Church
select
    names.[Value] as [Name of Thing],
    classes.[Value] as [Class of Thing]
from (
    select a.[Value],
           Appearance.value('@ThingUID', 'uniqueidentifier') as ThingUID
    from [v_Assertion] a
    cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "has name"]') T(Appearance)
) names
join (
    select a.[Value],
           DereferencingSet.value('Appearance[@Role = "thing"][1]/@ThingUID', 'uniqueidentifier') as ThingUID,
           DereferencingSet.value('Appearance[@Role = "class"][1]/@ThingUID', 'uniqueidentifier') as ClassUID
    from [v_Assertion] a
```

```
    cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet[Appearance/@Role = "class" and Appearance/@Role = "thing"]') T
(DereferencingSet)
) classifiers
on classifiers.ThingUID = names.ThingUID
join (
    select a.[Value],
           Appearance.value('@ThingUID', 'uniqueidentifier') as ClassUID
    from [v_Assertion] a
    cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') T(Appearance)
) classes
on classes.ClassUID = classifiers.ClassUID;

-- the Modeler can clarify the situation of Notre Dame having two classes
declare @ModelerUID24 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
);
declare @CathedralUID24 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Cathedral"]//*[@Role = "is class"]') as T(Appearance)
);
declare @ChurchUID24 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Church"]//*[@Role = "is class"]') as T(Appearance)
);
declare @assertion24 xml = '
    <Assertion PositorUID="' + @ModelerUID24 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
          <Appearance ThingUID="' + @CathedralUID24 + '" Role="subclass" />
          <Appearance ThingUID="' + @ChurchUID24 + '" Role="class" />
        </DereferencingSet>
        <Value>active</Value>
      </Posit>
    </Assertion>
';
exec Assert @assertion24 OUTPUT;
```

```

-- now Cathedral is a subclass of Church
select
    classes.[Value] as [Class],
    subclasses.[Value] as [Subclass]
from (
    select a.[Value],
           Appearance.value('@ThingUID', 'uniqueidentifier') as ClassUID
    from [v_Assertion] a
    cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') T(Appearance)
) classes
join (
    select a.[Value],
           DereferencingSet.value('Appearance[@Role = "subclass"][1]/@ThingUID', 'uniqueidentifier') as SubclassUID,
           DereferencingSet.value('Appearance[@Role = "class"][1]/@ThingUID', 'uniqueidentifier') as ClassUID
    from [v_Assertion] a
    cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet[Appearance/@Role = "class" and Appearance/@Role = "subclass"]') T
(DereferencingSet)
) inheritance
on inheritance.ClassUID = classes.ClassUID
join (
    select a.[Value],
           Appearance.value('@ThingUID', 'uniqueidentifier') as SubclassUID
    from [v_Assertion] a
    cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') T(Appearance)
) subclasses
on subclasses.SubclassUID = inheritance.SubclassUID;


/*
----------- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----------
def. of a posit type
A posit type, τ(p) = [{(C1,r1), . . . ,(Cn,rn)}, τ(v), τ(t)], for a
posit p = [{(i1,r1), . . . ,(in,rn)}, v, t], is a structure constructed
by replacing unique identifiers, ij with the unique identifiers of their
class, Cj, the value, v, with its data type, τ(v), and the time point, t,
with its data type, τ(t).
```

```
-----------------------------------------------------------------------------------------

A posit type is a structure, but posits in the relational model can only have
values of type varchar(max), so it cannot hold such a structure. We can, however,
store it as serialized XML, where the XML has the same structure as the one we
have been using to create posits.
*/

-- let the Modeler create a posit type for the ternary "marriage" relationship
declare @ModelerUID25 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
);
declare @PersonUID25 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Person"]//*[@Role = "is class"]') as T(Appearance)
);
declare @ChurchUID25 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Church"]//*[@Role = "is class"]') as T(Appearance)
);
declare @assertion25 xml = '
    <Assertion PositorUID="' + @ModelerUID25 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
          <Appearance ThingUID="T" Role="is type" />
        </DereferencingSet>
        <Value>
          <Posit AppearanceTime="datetime">
            <DereferencingSet>
              <Appearance ThingUID="' + @PersonUID25 + '" Role="husband" />
              <Appearance ThingUID="' + @PersonUID25 + '" Role="wife" />
              <Appearance ThingUID="' + @ChurchUID25 + '" Role="church" />
            </DereferencingSet>
            <Value>varchar(555)</Value>
          </Posit>
        </Value>
```

```
      </Posit>
    </Assertion>
';
exec Assert @assertion25 OUTPUT;

-- list the posit type
select a.* from [v_Assertion] a
where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is type"]') = 1
order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;

------------------------------------------------------ TODO BELOW ------------------------------------------------------------
-- let the Modeler also add the posit type for the unary 'has beard' relationship
declare @ModelerUID26 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
);
declare @PersonUID26 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Person"]//*[@Role = "is class"]') as T(Appearance)
);
declare @assertion26 xml = '
    <Assertion PositorUID="' + @ModelerUID26 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
      <Posit AppearanceTime="1998-12-31 23:59">
        <DereferencingSet>
          <Appearance ThingUID="T" Role="is type" />
        </DereferencingSet>
        <Value>
          <Posit AppearanceTime="datetime">
            <DereferencingSet>
              <Appearance ThingUID="' + @PersonUID26 + '" Role="has beard" />
            </DereferencingSet>
            <Value>varchar(555)</Value>
          </Posit>
        </Value>
      </Posit>
    </Assertion>
';
```

```
exec Assert @assertion26 OUTPUT;

-- list the posit types
select a.* from [v_Assertion] a
where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is type"]') = 1
order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;


-- now the Modeler can express that a class appears in a certain posit type
-- since this can be derived we can write some code that loops through posit types
-- (provided that they all appear "commonly" for the classes)
declare @ClassUID27 varchar(555);
declare @PositTypeUID27 varchar(555);

declare maps cursor static for (
    select distinct
        Appearance.value('@ThingUID', 'uniqueidentifier') as ClassUID,
        pt.PositTypeUID
    from (
        select cast(a.Value as xml) as PositType,
               AssertionXML.value('/Assertion[1]/Posit[1]/DereferencingSet[1]/Appearance[1]/@ThingUID', 'uniqueidentifier') as PositTypeUID
        from [v_Assertion] a
        where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is type"]') = 1
    ) pt
    cross apply PositType.nodes('/Posit/DereferencingSet/Appearance') as T(Appearance)
);

open maps;
fetch next from maps into @ClassUID27, @PositTypeUID27;

declare @ModelerUID27 varchar(555) = (
    select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
    cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
);

while(@@FETCH_STATUS = 0)
begin
```

```
    declare @assertion27 xml = '
        <Assertion PositorUID="' + @ModelerUID27 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
          <Posit AppearanceTime="1998-12-31 23:59">
            <DereferencingSet>
              <Appearance ThingUID="' + @ClassUID27 + '" Role="class" />
              <Appearance ThingUID="' + @PositTypeUID27 + '" Role="type" />
            </DereferencingSet>
            <Value>commonly</Value>
          </Posit>
        </Assertion>
    ';
    exec Assert @assertion27 OUTPUT;

    fetch next from maps into @ClassUID27, @PositTypeUID27;
end

close maps;
deallocate maps;

-- now the Church and Person classes are tied to the posit type for the ternary "marriage" relationship
-- and Person should also be tied to the unary "has beard" relationship
select
    classes.[Value] as [Class],
    classes.[ClassUID],
    cast(posit_types.[Value] as xml) as [PositType]
from (
    select a.[Value],
           Appearance.value('@ThingUID', 'uniqueidentifier') as ClassUID
    from [v_Assertion] a
    cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') T(Appearance)
) classes
join (
    select a.[Value],
           DereferencingSet.value('Appearance[@Role = "type"][1]/@ThingUID', 'uniqueidentifier') as PositTypeUID,
           DereferencingSet.value('Appearance[@Role = "class"][1]/@ThingUID', 'uniqueidentifier') as ClassUID
    from [v_Assertion] a
```

```
    cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet[Appearance/@Role = "class" and Appearance/@Role = "type"]') T
(DereferencingSet)
) typings
on typings.ClassUID = classes.ClassUID
join (
    select a.[Value],
           Appearance.value('@ThingUID', 'uniqueidentifier') as PositTypeUID
    from [v_Assertion] a
    cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is type"]') T(Appearance)
) posit_types
on posit_types.PositTypeUID = typings.PositTypeUID;

-- so, from the Modeler's point of view, a Person is that which can be married to a wife in a church and may have a beard
```