

```
1. use Transitional;
2. go
3. ----- RELATIONAL MODEL -----
4. --
5. -- Script by Lars Rönnbäck - latest version is always here: https://pastebin.com/AwdwCau8
6. --
7. --
8. -- Based on the paper "Modeling Conflicting, Unreliable, and Varying Information"
9. -- Full-text preprint available on ResearchGate: https://bit.ly/2A7J4Rb
10. --
11. -- This script creates a relationally modeled implementation of Transitional Modeling, with
12. -- some limitations due to missing features in SQL Server. Note that this is only intended
13. -- as a prototype in which concepts can be tested for educational purposes.
14. --
15. -- Version: 20181220.1 Added stored procedure that simplifies data insertion.
16. -- Cleaned up examples and added posit types.
17. -- 20181217.2 Better XML in the views (old version: https://pastebin.com/kuDmLwYq).
18. --
19. --
20. drop procedure if exists [Assert];
21. drop view if exists [Check_for_Contradictions];
22. drop function if exists [Information_in_Effect];
23. drop view if exists [v_Assertion];
24. drop view if exists [v_Posit];
25. drop table if exists [Assertion];
26. drop table if exists [Posit];
27. drop table if exists [Dereference];
28. drop table if exists [DereferencingSet];
29. drop table if exists [Appearance];
30. drop table if exists [Role];
31. drop table if exists [Thing];
32.
33. -- A table to store the unique identifiers of things.
34. -- Where a thing is that which is sufficiently distinguishable
35. -- from something else to be told apart.
36. create table [Thing] (
37.     [UID] uniqueidentifier not null default NEWSEQUENTIALID(),
```

```

38.     -- Enforce uniqueness of the unique identifiers.
39.     -- Note that primary keys enforce uniqueness as well as cluster
40.     -- the underlying table for better performance, and is needed
41.     -- in order to reference these using foreign key references.
42.     constraint [unique_and_referenceable_UID] primary key clustered (
43.         [UID]
44.     )
45. );
46.
47. create table [Role] (
48.     [RoleUID] uniqueidentifier not null,
49.     [Role] varchar(555) not null,
50.     constraint [Role_is_Thing] foreign key (
51.         [RoleUID]
52.     ) references [Thing]([UID]),
53.     constraint [referenceable_RoleUID] primary key clustered (
54.         [RoleUID]
55.     ),
56.     constraint [unique_Role] unique nonclustered (
57.         [Role]
58.     )
59. );
60.
61. /*
62. ----- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----
63. def. of universal
64. A body of information is said to be universal iff positors agree on all appearances.
65. -----
66.
67. In order to disagree you must first agree upon something upon which the difference in
68. opinion lies. At the very least, positors must agree on appearances. In other words
69. if two positors want to have an opinion about something, they must first come to the
70. conclusion that this something boils down to the same unique identifier for both of them
71. and that they mean the same when they talk about the roles it may appear in.
72.
73. We will assume that talk about "Archie's beard" by any positor means that it is the
74. same Archie and the same property they are talking about.

```

```

75. */
76. create table [Appearance] (
77.     [AppearanceUID] uniqueidentifier not null,
78.     [ThingUID] uniqueidentifier not null,
79.     [RoleUID] uniqueidentifier not null,
80.     constraint [Appearance_is_Thing] foreign key (
81.         [AppearanceUID]
82.     ) references [Thing]([UID]),
83.     constraint [existing_Thing] foreign key (
84.         [ThingUID]
85.     ) references [Thing]([UID]),
86.     constraint [existing_Role] foreign key (
87.         [RoleUID]
88.     ) references [Role]([RoleUID]),
89.     constraint [referenceable_AppearanceUID] primary key clustered (
90.         [AppearanceUID]
91.     ),
92.     constraint [unique_Appearance] unique nonclustered (
93.         [ThingUID],
94.         [RoleUID]
95.     )
96. );
97.
98. create table [DereferencingSet] (
99.     [DereferencingSetUID] uniqueidentifier not null,
100.    constraint [DereferencingSet_is_Thing] foreign key (
101.        [DereferencingSetUID]
102.    ) references [Thing]([UID]),
103.    constraint [unique_and_referenceable_DereferencingSetUID] primary key clustered (
104.        [DereferencingSetUID]
105.    )
106. );
107.
108. create table [Dereference] (
109.     [DereferencingSetUID] uniqueidentifier not null,
110.     [AppearanceUID] uniqueidentifier not null,
111.     constraint [reference_to_DereferencingSet] foreign key (

```

```

112.     [DereferencingSetUID]
113. ) references [DereferencingSet]([DereferencingSetUID]),
114. constraint [reference_to_Appearance] foreign key (
115.     [AppearanceUID]
116. ) references [Appearance]([AppearanceUID]),
117. constraint [unique_Dereference] primary key clustered (
118.     [DereferencingSetUID],
119.     [AppearanceUID]
120. )
121. );
122.
123. /*
124. ----- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----
125. def. of canonical
126. A body of information is said to be canonical iff all assertions
127. are made against posits without negated values.
128. -----
129.
130. In practice, the value of a posit may be of any data type, primitive or complex, but due to
131. SQL Server lacking support for generics (looks like <T> in many programming languages) we
132. have to limit the Value column to varchar(max). Appearance time may also be any time type,
133. exact or fuzzy, in order to represent that the value appeared exactly since some specified
134. time or inexactly within some period of time. Here the AppearanceTime column is limited
135. to a datetime.
136.
137. Finally, in order to represent information, canonical form is used, which simply means that
138. values are stored without negation, for example "red" is acceptable, and "not red" is not.
139. Opposite opinions are instead handled using negative Reliability in assertions.
140. */
141. create table [Posit] (
142.     [PositUID] uniqueidentifier not null,
143.     [DereferencingSetUID] uniqueidentifier not null,
144.     [Value] varchar(max) null,
145.     [AppearanceTime] datetime not null,
146.     constraint [existing_DereferencingSet] foreign key (
147.         [DereferencingSetUID]
148.     ) references [DereferencingSet]([DereferencingSetUID]),

```

```

149.     constraint [Posit_is_Thing] foreign key (
150.         [PositUID]
151.     ) references [Thing]([UID]),
152.     constraint [referenceable_PositUID] primary key clustered (
153.         [PositUID]
154.     )
155. );
156.
157. /*
158. ----- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----
159. def. of a body of information
160. A body of information is a set of true assertions.
161.
162. def. of exclusive
163. A body of information is said to be exclusive iff no assertions
164. in which only the reliability differs exist.
165. -----
166.
167. We will assume that any data inserted into this table represents true assertions. We will not
168. constrain the Reliability column, but in practice it should be in the interval [-1, 1].
169. The constraint [unique_Assertion] ensures that the stored assertions are exclusive, so that
170. there is no ambiguity with respect to the assigned Reliability.
171. */
172. create table [Assertion] (
173.     [AssertionUID] uniqueidentifier not null,
174.     [PositorUID] uniqueidentifier not null,
175.     [PositUID] uniqueidentifier not null,
176.     [Reliability] decimal(3,2) not null,
177.     [AssertionTime] datetime not null,
178.     constraint [Assertion_is_Thing] foreign key (
179.         [AssertionUID]
180.     ) references [Thing]([UID]),
181.     constraint [existing_Positor] foreign key (
182.         [PositorUID]
183.     ) references [Thing]([UID]),
184.     constraint [existing_Posit] foreign key (
185.         [PositUID]

```

```

186.     ) references [Posit]([PositUID]),
187.     constraint [referenceable_AssertionUID] primary key nonclustered (
188.         [AssertionUID]
189.     ),
190.     constraint [unique_Assertion] unique clustered (
191.         [PositorUID],
192.         [PositUID],
193.         [AssertionTime]
194.     )
195. );
196.
197. go
198. create or alter view [v_Posit]
199. as
200. select
201.     p.[PositUID],
202.     p.[DereferencingSetUID],
203.     p.[Value],
204.     p.[AppearanceTime],
205.     (
206.         select
207.             p.[PositUID] as [@PositUID],
208.             p.[AppearanceTime] as [@AppearanceTime],
209.             p.[DereferencingSetUID] as [DereferencingSet/@DereferencingSetUID],
210.             (
211.                 select
212.                     a.[AppearanceUID] as [@AppearanceUID],
213.                     a.[ThingUID] as [@ThingUID],
214.                     r.[Role] as [@Role]
215.                 from
216.                     [DereferencingSet] s
217.                 join
218.                     [Dereference] d
219.                 on
220.                     d.[DereferencingSetUID] = s.[DereferencingSetUID]
221.                 join
222.                     [Appearance] a

```

```

223.         on
224.             a.[AppearanceUID] = d.[AppearanceUID]
225.         join
226.             [Role] r
227.         on
228.             r.[RoleUID] = a.[RoleUID]
229.         where
230.             s.[DereferencingSetUID] = p.[DereferencingSetUID]
231.         order by
232.             r.[RoleUID] asc
233.         for xml path('Appearance'), type
234.             ) as [DereferencingSet],
235.         p.[Value]
236.         for xml path('Posit'), type
237.             ) as [PositXML]
238. from
239.     [Posit] p
240.
241. go
242. create or alter view [v_Assertion]
243. as
244. select
245.     a.[AssertionUID],
246.     a.[PositorUID],
247.     a.[PositUID],
248.     p.[DereferencingSetUID],
249.     p.[Value],
250.     p.[AppearanceTime],
251.     a.[Reliability],
252.     a.[AssertionTime],
253.     (
254.         select
255.             a.[AssertionUID] as [@AssertionUID],
256.             a.[PositorUID] as [@PositorUID],
257.             a.[Reliability] as [@Reliability],
258.             a.[AssertionTime] as [@AssertionTime],
259.             p.[PositXML].query('.')

```

```

260.     for xml path('Assertion'), type
261.   ) as [AssertionXML]
262. from
263.   [Assertion] a
264. join
265.   [v_Posit] p
266. on
267.   p.[PositUID] = a.[PositUID]
268. go
269.
270. /*
271. ----- XML FORMAT -----
272. Specify UID for already known things and some Label for unknown. Bracketed attributes
273. are optional. Labels will result in new things being created, one for each unique label.
274.
275.   <Assertion PositorUID="<UID|Label>" Reliability="1.00" AssertionTime="2018-12-13T15:30:00">
276.     <Posit [PositUID="<UID>"] AppearanceTime="2018-12-01T12:00:00">
277.       <DereferencingSet [DereferencingSetUID="<UID>"]>
278.         <Appearance [AppearanceUID="<UID>"] ThingUID="<UID|Label>" Role="has beard" />
279.       </DereferencingSet>
280.       <Value>fluffy red</Value>
281.     </Posit>
282.   </Assertion>
283.
284. Note that AssertionUID cannot be specified. Assertions can never be altered once they've been
285. created.
286. -----
287. */
288. create or alter procedure Assert (
289.   @assertion xml OUTPUT
290. )
291. as
292. begin
293.   set nocount on;
294.
295.   -- create a lookup table
296.   create table #Thing (

```



```

297.     [ThingUID] uniqueidentifier not null primary key,
298.     [UID_or_Label] varchar(555) not null
299. );
300.
301. -- is a posit set already referenced through a UID?
302. declare @PositUID uniqueidentifier = @assertion.value(
303.     '/Assertion[1]/Posit[1]/@PositUID',
304.     'uniqueidentifier'
305. );
306.
307. -- if not it needs to be built
308. if @PositUID is null
309. begin
310.     -- is a dereferencing set already referenced through a UID?
311.     declare @DereferencingSetUID uniqueidentifier = @assertion.value(
312.         '/Assertion[1]/Posit[1]/DereferencingSet[1]/@DereferencingSetUID',
313.         'uniqueidentifier'
314.     );
315.
316.     if @DereferencingSetUID is not null
317.     begin
318.         if not exists (
319.             select [DereferencingSetUID] from [DereferencingSet]
320.             where [DereferencingSetUID] = @DereferencingSetUID
321.         )
322.             raiserror('The referenced @DereferencingSetUID does not exist', 10, 1);
323.     end
324.     else
325.     begin
326.         -- check if given AppearanceUID:s exist
327.         if exists (
328.             select x.[AppearanceUID]
329.             from (
330.                 select distinct
331.                     Appearance.value('@AppearanceUID', 'uniqueidentifier') as [AppearanceUID]
332.                 from @assertion.nodes(
333.                     '/Assertion/Posit/DereferencingSet/Appearance[@AppearanceUID]'

```

```

334.         ) as T(Appearance)
335.     ) x
336.     left join [Appearance] a
337.     on a.[AppearanceUID] = x.[AppearanceUID]
338.     where a.[AppearanceUID] is null
339. )
340. raiserror('The referenced @AppearanceUID does not exist', 10, 1);
341.
342. -- create a table to hold appearances
343. create table #Appearance (
344.     AppearanceUID uniqueidentifier not null primary key,
345.     ThingUID uniqueidentifier not null,
346.     RoleUID uniqueidentifier not null
347. );
348.
349. -- add existing and referenced appearances
350. insert into #Appearance
351. select a.*
352. from (
353.     select distinct
354.         Appearance.value('@AppearanceUID', 'uniqueidentifier') as [AppearanceUID]
355.     from @assertion.nodes(
356.         '/Assertion/Posit/DereferencingSet/Appearance[@AppearanceUID]'
357.     ) as T(Appearance)
358. ) x
359. join [Appearance] a
360. on a.[AppearanceUID] = x.[AppearanceUID];
361.
362. declare @matches int = 0;
363. -- create missing things and populate the lookup table
364. merge [Thing] t
365. using (
366.     select distinct
367.         Appearance.value('@ThingUID', 'varchar(555)') as [UID_or_Label]
368.     from @assertion.nodes(
369.         '/Assertion/Posit/DereferencingSet/Appearance[not(@AppearanceUID)]'
370.     ) as T(Appearance)

```

```

371.         ) i on i.[UID_or_Label] = cast(t.[UID] as varchar(555))
372.         when not matched then insert values(DEFAULT)
373.         when matched then update set @matches = @matches + 1
374.         output inserted.[UID], i.[UID_or_Label] into #Thing;
375.
376.         -- create a lookup table
377.         create table #Role (
378.             [RoleUID] uniqueidentifier not null primary key,
379.             [Role] varchar(555) not null
380.         );
381.
382.         -- lookup or create the roles
383.         merge [Thing] t
384.         using (
385.             select
386.                 r.[RoleUID],
387.                 x.[Role]
388.             from (
389.                 select
390.                     Appearance.value('@Role', 'varchar(555)') as [Role]
391.                 from @assertion.nodes(
392.                     '/Assertion/Posit/DereferencingSet/Appearance[not(@AppearanceUID)]'
393.                 ) as T(Appearance)
394.             ) x
395.             left join [Role] r
396.             on x.[Role] = r.[Role]
397.         ) i on i.[RoleUID] = t.[UID]
398.         when not matched then insert values(DEFAULT)
399.         when matched then update set @matches = @matches + 1
400.         output inserted.[UID], i.[Role] into #Role;
401.
402.         -- add missing roles to Role
403.         merge [Role] r
404.         using #Role r1kp
405.         on r1kp.[RoleUID] = r.[RoleUID]
406.         when not matched then insert values (r1kp.[RoleUID], r1kp.[Role]);
407.

```

```

408.      -- lookup or create the appearances
409.      merge [Thing] t
410.      using (
411.          select
412.              a.[AppearanceUID],
413.              tlpk.[ThingUID],
414.              rlpk.[RoleUID]
415.          from (
416.              select
417.                  Appearance.value('@ThingUID', 'varchar(555)') as [UID_or_Label],
418.                  Appearance.value('@Role', 'varchar(555)') as [Role]
419.              from @assertion.nodes(
420.                  '/Assertion/Posit/DereferencingSet/Appearance[not(@AppearanceUID)]'
421.              ) as T(Appearance)
422.          ) x
423.          join #Thing tlpk
424.          on tlpk.[UID_or_Label] = x.[UID_or_Label]
425.          join #Role rlpk
426.          on rlpk.[Role] = x.[Role]
427.          left join [Appearance] a
428.          on a.[ThingUID] = tlpk.[ThingUID] and a.[RoleUID] = rlpk.[RoleUID]
429.      ) i on i.[AppearanceUID] = t.[UID]
430.      when not matched then insert values(DEFAULT)
431.      when matched then update set @matches = @matches + 1
432.      output inserted.[UID], i.[ThingUID], i.[RoleUID] into #Appearance;
433.
434.      -- add missing apperances to Appearance
435.      merge [Appearance] a
436.      using #Appearance alkp
437.      on alkp.[AppearanceUID] = a.[AppearanceUID]
438.      when not matched then insert values (alkp.[AppearanceUID], alkp.[ThingUID], alkp.[RoleUID]);
439.
440.      declare @NumberOfAppearances int = (select count(*) from #Appearance);
441.
442.      -- do these appearances already exist as a dereferencing set?
443.      set @DereferencingSetUID = (
444.          select d.DereferencingSetUID

```

```

445.         from [Dereference] d
446.     left join #Appearance a on a.AppearanceUID = d.AppearanceUID
447.     group by d.DereferencingSetUID
448.     having COUNT(d.AppearanceUID) = @NumberOfAppearances
449.         and COUNT(a.AppearanceUID) = @NumberOfAppearances
450. );
451.
452. -- if not it needs to be created
453. if @DereferencingSetUID is null
454. begin
455.     create table #DereferencingSet (
456.         [DereferencingSetUID] uniqueidentifier not null primary key
457.     );
458.
459.     insert into [Thing]
460.     output inserted.[UID] into #DereferencingSet
461.     values (DEFAULT);
462.
463.     set @DereferencingSetUID = (select top 1 [DereferencingSetUID] from #DereferencingSet);
464.
465.     insert into [DereferencingSet] values (@DereferencingSetUID);
466.
467.     insert into [Dereference]
468.     select @DereferencingSetUID, [AppearanceUID]
469.     from #Appearance;
470. end
471. end -- end of building a dereference set
472. -- now we have a @DereferencingSetUID
473.
474. declare @Value varchar(max);
475. declare @AppearanceTime datetime;
476.
477. select
478.     @Value = cast(Posit.query('Value/node()') as varchar(max)),
479.     @AppearanceTime = Posit.value('@AppearanceTime', 'datetime')
480. from @assertion.nodes(
481.     '/Assertion[1]/Posit[1]'

```

```
482.     ) as T(Posit);
483.
484.     -- check if it already exists
485.     set @PositUID = (
486.         select [PositUID]
487.         from [Posit]
488.         where [DereferencingSetUID] = @DereferencingSetUID
489.             and [Value] = @Value
490.             and [AppearanceTime] = @AppearanceTime
491.     );
492.
493.     if @PositUID is null
494.     begin
495.         -- create the posit
496.         create table #Posit (
497.             [PositUID] uniqueidentifier not null primary key
498.         );
499.
500.         insert into [Thing]
501.         output inserted.[UID] into #Posit
502.         values (DEFAULT);
503.
504.         set @PositUID = (select top 1 [PositUID] from #Posit);
505.
506.         insert into [Posit] (
507.             [PositUID],
508.             [DereferencingSetUID],
509.             [Value],
510.             [AppearanceTime]
511.         )
512.         values (
513.             @PositUID,
514.             @DereferencingSetUID,
515.             @Value,
516.             @AppearanceTime
517.         );
518.     end -- new posit created
```

```
519.     end -- end of building posit
520.     -- now we have a @PositUID
521.
522.     declare @AssertionUID uniqueidentifier;
523.
524.     -- create the assertion
525.     create table #Assertion (
526.         [AssertionUID] uniqueidentifier not null primary key
527.     );
528.
529.     insert into [Thing]
530.     output inserted.[UID] into #Assertion
531.     values (DEFAULT);
532.
533.     set @AssertionUID = (select top 1 [AssertionUID] from #Assertion);
534.
535.     -- A positor must be referenced through a UID or Label
536.     declare @PositorUID_or_Label varchar(555) = @assertion.value(
537.         '/Assertion[1]/@PositorUID',
538.         'varchar(555)'
539.     );
540.
541.     if @PositorUID_or_Label is null
542.     raiserror('No @PositorUID has been specified', 10, 1);
543.
544.     -- is must either be a UID or a Label (and if it is a Label it is already created)
545.     declare @PositorUID uniqueidentifier = isnull(
546.         (select [ThingUID] from #Thing where [UID_or_Label] = @PositorUID_or_Label),
547.         (select [UID] from [Thing] where cast([UID] as varchar(555)) = @PositorUID_or_Label)
548.     );
549.
550.     if @PositorUID is null
551.     raiserror('The referenced @PositorUID does not exist', 10, 1);
552.
553.     -- now store the assertion
554.     insert into [Assertion] (
555.         [AssertionUID],
```

```

556.     [PositorUID],
557.     [PositUID],
558.     [Reliability],
559.     [AssertionTime]
560. )
561. select
562.     @AssertionUID,
563.     @PositorUID,
564.     @PositUID,
565.     Posit.value('@Reliability', 'decimal(3,2)') as [AppearanceTime],
566.     Posit.value('@AssertionTime', 'datetime') as [AppearanceTime]
567. from @assertion.nodes(
568.     '/Assertion[1]'
569. ) as T(Posit)
570.
571. -- and we are done, so finally set the output variable
572. set @assertion = (
573.     select [AssertionXML] from [v_Assertion] where [AssertionUID] = @AssertionUID
574. );
575. end
576. go
577.
578. ----- PROPERTIES -----
579. -- first create a few things (and positors), and relate them to a name using "has name"
580. declare @assertion1 xml = '
581.     <Assertion PositorUID="A" Reliability="1.00" AssertionTime="2018-12-15 00:00">
582.         <Posit AppearanceTime="1972-08-20 15:30">
583.             <DereferencingSet>
584.                 <Appearance ThingUID="A" Role="has name" />
585.             </DereferencingSet>
586.             <Value>Archie</Value>
587.         </Posit>
588.     </Assertion>
589. ';
590. exec Assert @assertion1 OUTPUT;
591.
592. -- the stored assertion is returned (in case we need any of the UID:s for later)

```



```
593. select @assertion1;
594. -- but we can also look at it using the view
595. select * from [v_Assertion];
596.
597. declare @assertion2 xml = '
598.     <Assertion PositorUID="B" Reliability="1.00" AssertionTime="2018-12-15 00:00">
599.         <Posit AppearanceTime="1972-02-13 08:00">
600.             <DereferencingSet>
601.                 <Appearance ThingUID="B" Role="has name" />
602.             </DereferencingSet>
603.             <Value>Bella</Value>
604.         </Posit>
605.     </Assertion>
606. ';
607. exec Assert @assertion2 OUTPUT;
608.
609. declare @assertion3 xml = '
610.     <Assertion PositorUID="M" Reliability="1.00" AssertionTime="2018-12-15 00:00">
611.         <Posit AppearanceTime="1945-10-17 12:00">
612.             <DereferencingSet>
613.                 <Appearance ThingUID="M" Role="has name" />
614.             </DereferencingSet>
615.             <Value>Modeler</Value>
616.         </Posit>
617.     </Assertion>
618. ';
619. exec Assert @assertion3 OUTPUT;
620.
621. declare @assertion4 xml = '
622.     <Assertion PositorUID="S" Reliability="1.00" AssertionTime="2018-12-15 00:00">
623.         <Posit AppearanceTime="2018-12-15">
624.             <DereferencingSet>
625.                 <Appearance ThingUID="S" Role="has name" />
626.             </DereferencingSet>
627.             <Value>Script</Value>
628.         </Posit>
629.     </Assertion>
```

```

630. ';
631. exec Assert @assertion4 OUTPUT;
632.
633. declare @assertion5 xml = '
634.     <Assertion PositorUID="D" Reliability="1.00" AssertionTime="2018-12-15 00:00">
635.         <Posit AppearanceTime="2001-01-01 01:00">
636.             <DereferencingSet>
637.                 <Appearance ThingUID="D" Role="has name" />
638.             </DereferencingSet>
639.             <Value>Disser</Value>
640.         </Posit>
641.     </Assertion>
642. ';
643. exec Assert @assertion5 OUTPUT;
644.
645. -- there should now be five assertions
646. select * from [v_Assertion];
647.
648. -- first we will let Bella assert a few things about Archie
649. declare @ArchieUID6 varchar(555) = (
650.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
651.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
652. );
653. declare @BellaUID6 varchar(555) = (
654.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
655.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
656. );
657. declare @assertion6 xml = '
658.     <Assertion PositorUID="' + @BellaUID6 + '" Reliability="1.00" AssertionTime="2018-12-13 15:30">
659.         <Posit AppearanceTime="2018-12-01 12:00">
660.             <DereferencingSet>
661.                 <Appearance ThingUID="' + @ArchieUID6 + '" Role="has beard" />
662.             </DereferencingSet>
663.             <Value>fluffy red</Value>
664.         </Posit>
665.     </Assertion>
666. ';

```

```

667. exec Assert @assertion6 OUTPUT;
668.
669. -- there should now be a new assertion
670. select * from [v_Assertion] where [Value] = 'fluffy red';
671.
672. -- let the Disser have a different opinion
673. declare @ArchieUID7 varchar(555) = (
674.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
675.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
676. );
677. declare @DisserUID7 varchar(555) = (
678.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
679.     cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
680. );
681. declare @assertion7 xml = '
682.     <Assertion PositorUID="" + @DisserUID7 + "" Reliability="1.00" AssertionTime="2018-12-13 15:30">
683.         <Posit AppearanceTime="2018-12-01 12:00">
684.             <DereferencingSet>
685.                 <Appearance ThingUID="" + @ArchieUID7 + "" Role="has beard" />
686.             </DereferencingSet>
687.             <Value>shaved clean</Value>
688.         </Posit>
689.     </Assertion>
690. ';
691. exec Assert @assertion7 OUTPUT;
692.
693. -- there should now be two assertions
694. -- where Bella and the Disagreer have conflicting views of Archie's beard
695. select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean');
696.
697. -- let the Disser further oppose to what Bella stated
698. declare @ArchieUID8 varchar(555) = (
699.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
700.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
701. );
702. declare @DisserUID8 varchar(555) = (
703.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion

```

```

704.     cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
705. );
706. declare @assertion8 xml = '
707.     <Assertion PositorUID="' + @DisserUID8 + '" Reliability="-1.00" AssertionTime="2018-12-13 15:35">
708.         <Posit AppearanceTime="2018-12-01 12:00">
709.             <DereferencingSet>
710.                 <Appearance ThingUID="' + @ArchieUID8 + '" Role="has beard" />
711.             </DereferencingSet>
712.             <Value>fluffy red</Value>
713.         </Posit>
714.     </Assertion>
715. ';
716. exec Assert @assertion8 OUTPUT;
717.
718. -- so Bella and the Disser have opposing views of the posit stating it is fluffy red
719. select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean');
720.
721. -- let Bella assert that there is a very small chance it actually was shaved clean
722. declare @ArchieUID9 varchar(555) = (
723.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
724.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
725. );
726. declare @BellaUID9 varchar(555) = (
727.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
728.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
729. );
730. declare @assertion9 xml = '
731.     <Assertion PositorUID="' + @BellaUID9 + '" Reliability="0.05" AssertionTime="2018-12-13 15:35">
732.         <Posit AppearanceTime="2018-12-01 12:00">
733.             <DereferencingSet>
734.                 <Appearance ThingUID="' + @ArchieUID9 + '" Role="has beard" />
735.             </DereferencingSet>
736.             <Value>shaved clean</Value>
737.         </Posit>
738.     </Assertion>
739. ';
740. 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 occurred 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  $A(T@, t@) \subset A$  given a bitemporal point in assertion
776. and appearance time  $(T@, t@) \in T \times t$ . Assuming  $P, \alpha, T, D, v$ ,
777. and  $t$  are free variables, all assertions  $!(P, p, \alpha, T) \in A(T@, t@)$ 

```

```

778. with p = [D, v, t] are found using the following selection
779. criteria:
780. 1. Let A1 ⊂ A be the assertions in A satisfying T ≤ T@ and
781.    t ≤ t@.
782. 2. Let A2 ⊂ A1 be the assertions in A1 with α ≠ 0 and the
783.    latest appearance time t for each combination of P and
784.    D, excluding those assertions !(P, p, α, T) for which an
785.    assertion !(P, p, 0, T0) exists with T ≤ T0 ≤ T@.
786. 3. Let A(T@, t@) ⊂ A2 be the assertions from A2 with the
787.    latest assertion time T for each combination of P, D, and v.
788. -----
789.
790. A function for the information in effect is created below according
791. to the selection criteria defined above.
792. */
793. go
794. create or alter function [Information_in_Effect] (
795.     @appearanceTime datetime,
796.     @assertionTime datetime
797. )
798. returns table as return
799. with A1 as (
800.     select *
801.     from [v_Assertion]
802.     where [AppearanceTime] <= @appearanceTime
803.     and [AssertionTime] <= @assertionTime
804. ),
805. A2 as (
806.     select a.*
807.     from A1 a
808.     where a.[Reliability] <> 0
809.     and a.[AppearanceTime] = (
810.         select max(s.[AppearanceTime])
811.         from A1 s
812.         where s.[PositorUID] = a.[PositorUID]
813.         and s.[DereferencingSetUID] = a.[DereferencingSetUID]
814.     )

```

```

815.     and not exists (
816.         select x.[AssertionUID]
817.         from A1 x
818.         where x.[PositorUID] = a.[PositorUID]
819.             and x.[PositUID] = a.[PositUID]
820.             and x.[Reliability] = 0
821.             and x.[AssertionTime] between a.AssertionTime and @assertionTime
822.     )
823. )
824. select a.*
825.     from A2 a
826.     where a.[AssertionTime] = (
827.         select max(s.[AssertionTime])
828.         from A2 s
829.         where s.[PositorUID] = a.[PositorUID]
830.             and s.[DereferencingSetUID] = a.[DereferencingSetUID]
831.             and s.[Value] = a.[Value]
832.     )
833. go
834.
835. -- which information is in effect at 12 and 13 given what was asserted on or before 15:30?
836. select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:30')
837. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
838. select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:30')
839. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
840.
841. -- which information is in effect at 12 and 13 given what was asserted on or before 15:35?
842. select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:35')
843. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
844. select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:35')
845. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
846.
847. /*
848. ----- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----
849. def. of symmetric
850. A body of information is said to be symmetric iff the assertions
851.  $!(P, p, -\alpha, T)$  and  $!(P, \bar{p}, \alpha, T)$  are equivalent for a posit  $p$  and its opposite  $\bar{p}$ .

```

```

852.
853. def. of bounded
854. A body of information is said to be bounded iff the reliabilities in the assertions
855.  $!(P, p, \alpha, T)$  and  $!(P, \bar{p}, \beta, T)$  satisfy  $|\alpha + \beta| = |\alpha + \beta|^2$ .
856.
857. def. of non-contradictory
858. A body of information is said to be non-contradictory iff for
859. every information in effect the reliabilities, numbered 1 to n,
860. in all positive and negative assertions made by the same positor
861. for the same dereferencing set satisfy the inequality:
862. 
$$\frac{1}{2} \sum_{i=1}^n (1 - \text{sign}(\alpha_i)) + \sum_{i=1}^n \alpha_i \leq 1$$

863.
864. -----
865.
866.
867. The fact that we can talk about contradictions relies on the assumption that our body
868. of information is symmetric. When a body of information is also bounded it is possible to
869. transform all information into canonical form (negated values like "not red" are instead
870. expressed using "red" and a negative Reliability).
871. */
872.
873. -- we create a view that checks for contradictions according to the inequality above
874. go
875. create or alter view [Check_for_Contradictions]
876. as
877. with bitemporalTimepoints as (
878.     select [AppearanceTime], [AssertionTime]
879.     from (select distinct [AppearanceTime] from Posit) p
880.     cross join (select distinct [AssertionTime] from Assertion) a
881. )
882. select
883.     ineq.*,
884.     case
885.         when InequalitySum <= 1 then 'Non-Contradictory'
886.         else 'Contradictory'
887.     end as Result
888. from (

```



```

889.     select
890.         t.[AppearanceTime],
891.         t.[AssertionTime],
892.         iie.[PositorUID],
893.         iie.[DereferencingSetUID],
894.         0.5 * sum(1 - sign(iie.[Reliability])) + sum(iie.[Reliability]) as InequalitySum
895.     from
896.         bitemporalTimepoints t
897.     cross apply
898.         [Information_in_Effect](t.[AppearanceTime], t.[AssertionTime]) iie
899.     group by
900.         t.[AppearanceTime],
901.         t.[AssertionTime],
902.         iie.[PositorUID],
903.         iie.[DereferencingSetUID]
904. ) ineq;
905. go
906.
907. -- have contradictory assertions been made?
908. select * from [Check_for_Contradictions];
909. -- Bella is contradictory for the appearance time between 12 and 13 for assertion time 15:35 and onwards
910. -- so with our current knowledge she has made contradictory statements
911.
912. -- Bella realizes she will remain forever contradictory unless she makes another assertion
913. -- Bella reevaluates her earlier assertion from 1 to 0.95
914. declare @ArchieUID11 varchar(555) = (
915.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
916.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
917. );
918. declare @BellaUID11 varchar(555) = (
919.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
920.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
921. );
922. declare @assertion11 xml = '
923.     <Assertion PositorUID="' + @BellaUID11 + '" Reliability="0.95" AssertionTime="2018-12-13 15:40">
924.         <Posit AppearanceTime="2018-12-01 12:00">
925.             <DereferencingSet>

```

```

926.         <Appearance ThingUID="" + @ArchieUID11 + "" Role="has beard" />
927.     </DereferencingSet>
928.     <Value>fluffy red</Value>
929. </Posit>
930. </Assertion>
931. ';
932. exec Assert @assertion11 OUTPUT;
933.
934. -- which information is in effect at 12 and 13 given that assertion
935. select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:40')
936. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
937. select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:40')
938. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
939. -- now the situation looks better for Bella between 12 and 13
940.
941. -- have contradictory assertions been made?
942. select * from [Check_for_Contradictions];
943. -- Bella is contradictory for the appearance time between 12 and 13 but now only for
944. -- assertion time between 15:35 and 15:45
945.
946. -- the Disagreer is now changing it's mind and retracts the previous statement
947. -- a retraction has reliability = 0
948. declare @ArchieUID12 varchar(555) = (
949.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
950.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
951. );
952. declare @DisserUID12 varchar(555) = (
953.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
954.     cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
955. );
956. declare @assertion12 xml = '
957.     <Assertion PositorUID="" + @DisserUID12 + "" Reliability="0" AssertionTime="2018-12-13 15:40">
958.         <Posit AppearanceTime="2018-12-01 12:00">
959.             <DereferencingSet>
960.                 <Appearance ThingUID="" + @ArchieUID12 + "" Role="has beard" />
961.             </DereferencingSet>
962.             <Value>shaved clean</Value>

```

```

963.     </Posit>
964. </Assertion>
965. ';
966. exec Assert @assertion12 OUTPUT;
967.
968. -- which information is now in effect at 12 and 13 at the time of the retraction?
969. select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:40')
970. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
971. select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:40')
972. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
973. -- the Disser is now only saying it's not fluffy red
974.
975. -- Bella is going to reiterate that Archie's beard was also shaved clean one hour later
976. declare @ArchieUID13 varchar(555) = (
977.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
978.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
979. );
980. declare @BellaUID13 varchar(555) = (
981.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
982.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
983. );
984. declare @assertion13 xml = '
985.     <Assertion PositorUID="' + @BellaUID13 + '" Reliability="1.00" AssertionTime="2018-12-13 15:45">
986.         <Posit AppearanceTime="2018-12-01 14:00">
987.             <DereferencingSet>
988.                 <Appearance ThingUID="' + @ArchieUID13 + '" Role="has beard" />
989.             </DereferencingSet>
990.             <Value>shaved clean</Value>
991.         </Posit>
992.     </Assertion>
993. ';
994. exec Assert @assertion13 OUTPUT;
995.
996. -- the latest assertion is called a "restatement", since the value is not changing
997. select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean')
998. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
999.

```

```

1000. -- the information in effect will show the latest restatement
1001. select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:45')
1002. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1003. select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:45')
1004. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1005. select * from [Information_in_Effect]('2018-12-01 14:00', '2018-12-13 15:45')
1006. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1007.
1008. -- Bella is going to reassure that she still is certain Archie's beard was shaved clean at 14
1009. declare @ArchieUID14 varchar(555) = (
1010.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1011.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
1012. );
1013. declare @BellaUID14 varchar(555) = (
1014.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1015.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
1016. );
1017. declare @assertion14 xml = '
1018.     <Assertion PositorUID="' + @BellaUID14 + '" Reliability="1.00" AssertionTime="2018-12-13 15:50">
1019.         <Posit AppearanceTime="2018-12-01 14:00">
1020.             <DereferencingSet>
1021.                 <Appearance ThingUID="' + @ArchieUID14 + '" Role="has beard" />
1022.             </DereferencingSet>
1023.             <Value>shaved clean</Value>
1024.         </Posit>
1025.     </Assertion>
1026. ';
1027. exec Assert @assertion14 OUTPUT;
1028.
1029. -- the latest assertion is called a "reassertion", since only the assertion time changes
1030. select * from [v_Assertion] where [Value] in ('fluffy red', 'shaved clean')
1031. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1032.
1033. -- the information in effect will show the latest restatement and latest reassertion
1034. select * from [Information_in_Effect]('2018-12-01 12:00', '2018-12-13 15:45')
1035. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1036. select * from [Information_in_Effect]('2018-12-01 13:00', '2018-12-13 15:45')

```

```

1037. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1038. select * from [Information_in_Effect]('2018-12-01 14:00', '2018-12-13 15:45')
1039. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1040. select * from [Information_in_Effect]('2018-12-01 14:00', '2018-12-13 15:50')
1041. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1042.
1043. ----- METADATA -----
1044. -- The good thing about posits, assertions, and so on being "things" in themselves
1045. -- is that it makes it possible to produce posits about them as well. This is
1046. -- usually called metadata.
1047.
1048. -- let the Script positor assert when what we have done so far was recorded in
1049. -- our database right now.
1050.
1051. -- we now have 42 things that we want to create metadata for
1052. select * from [Thing];
1053.
1054. declare @ScriptUID15 varchar(555) = (
1055.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1056.     cross apply AssertionXML.nodes('//*[Value = "Script"]//*[@Role = "has name"]') as T(Appearance)
1057. );
1058.
1059. declare thing cursor static for (select [UID] from [Thing]);
1060. open thing;
1061.
1062. declare @ThingUID15 varchar(555);
1063. fetch next from thing into @ThingUID15;
1064.
1065. declare @now char(19) = CONVERT(varchar(19), getdate(), 121);
1066.
1067. while(@@FETCH_STATUS = 0)
1068. begin
1069.     declare @assertion15 xml = '
1070.         <Assertion PositorUID="' + @ScriptUID15 + '" Reliability="1.00" AssertionTime="' + @now + '">
1071.             <Posit AppearanceTime="' + @now + '">
1072.                 <DereferencingSet>
1073.                     <Appearance ThingUID="' + @ThingUID15 + '" Role="was recorded at" />

```

```

1074.         </DereferencingSet>
1075.         <Value>' + @now + '</Value>
1076.     </Posit>
1077. </Assertion>
1078. ';
1079. exec Assert @assertion15 OUTPUT;
1080.
1081.     fetch next from thing into @ThingUID15;
1082. end
1083.
1084. close thing;
1085. deallocate thing;
1086.
1087. -- now there are 42 assertions from which we can tell when the data entered the database
1088. select * from [v_Assertion] where isdate([Value]) = 1
1089. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1090.
1091. -- note that the metadata is not in effect if we travel back in time
1092. select * from [Information_in_Effect]('2018-12-01 14:00', '2018-12-13 15:50')
1093. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1094.
1095. -- but it is in effect now
1096. select * from [Information_in_Effect](getdate(), getdate())
1097. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1098.
1099. ----- RELATIONSHIPS -----
1100. -- we will start by marrying Archie and Bella in the Notre-Dame
1101. -- so we first need the Notre-Dame thing
1102. declare @assertion16 xml = '
1103.     <Assertion PositorUID="NDame" Reliability="1.00" AssertionTime="2018-12-15 00:00">
1104.         <Posit AppearanceTime="1753-01-01">
1105.             <DereferencingSet>
1106.                 <Appearance ThingUID="NDame" Role="has name" />
1107.             </DereferencingSet>
1108.             <Value>Notre-Dame de Paris</Value>
1109.         </Posit>
1110.     </Assertion>

```

```

1111. ';
1112. exec Assert @assertion16 OUTPUT;
1113.
1114. -- we need the UID:s
1115. declare @ArchieUID17 varchar(555) = (
1116.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1117.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
1118. );
1119. declare @BellaUID17 varchar(555) = (
1120.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1121.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
1122. );
1123. declare @NDameUID17 varchar(555) = (
1124.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1125.     cross apply AssertionXML.nodes('//*[Value = "Notre-Dame de Paris"]//*[@Role = "has name"]') as T(Appearance)
1126. );
1127. declare @assertion17 xml = '
1128.     <Assertion PositorUID="' + @BellaUID17 + '" Reliability="1.00" AssertionTime="2018-12-13 15:55">
1129.         <Posit AppearanceTime="2004-06-19 15:00">
1130.             <DereferencingSet>
1131.                 <Appearance ThingUID="' + @ArchieUID17 + '" Role="husband" />
1132.                 <Appearance ThingUID="' + @BellaUID17 + '" Role="wife" />
1133.                 <Appearance ThingUID="' + @NDameUID17 + '" Role="church" />
1134.             </DereferencingSet>
1135.             <Value>married</Value>
1136.         </Posit>
1137.     </Assertion>
1138. ';
1139. exec Assert @assertion17 OUTPUT;
1140.
1141. -- now Bella has asserted the marriage
1142. select * from [v_Assertion] where [Value] in ('married')
1143. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1144.
1145. -- let them divorce a while later (using the same dereferencing set, so it is a real change)
1146. declare @ArchieUID18 varchar(555) = (
1147.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion

```

```

1148.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
1149. );
1150. declare @BellaUID18 varchar(555) = (
1151.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1152.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
1153. );
1154. declare @NDameUID18 varchar(555) = (
1155.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1156.     cross apply AssertionXML.nodes('//*[Value = "Notre-Dame de Paris"]//*[@Role = "has name"]') as T(Appearance)
1157. );
1158. declare @assertion18 xml = '
1159.     <Assertion PositorUID="' + @BellaUID18 + '" Reliability="1.00" AssertionTime="2018-12-13 15:55">
1160.         <Posit AppearanceTime="2010-01-31 10:00">
1161.             <DereferencingSet>
1162.                 <Appearance ThingUID="' + @ArchieUID18 + '" Role="husband" />
1163.                 <Appearance ThingUID="' + @BellaUID18 + '" Role="wife" />
1164.                 <Appearance ThingUID="' + @NDameUID18 + '" Role="church" />
1165.             </DereferencingSet>
1166.             <Value>divorced</Value>
1167.         </Posit>
1168.     </Assertion>
1169. ';
1170. exec Assert @assertion18 OUTPUT;
1171.
1172. -- now Bella has asserted the divorce as well
1173. select * from [v_Assertion] where [Value] in ('married', 'divorced')
1174. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1175.
1176. -- however, as it turns out, Bella remarried the Disagreer at a later time, but not in a church
1177. -- that means a new dereferencing set is needed and this time without the church
1178. declare @DisserUID19 varchar(555) = (
1179.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1180.     cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
1181. );
1182. declare @BellaUID19 varchar(555) = (
1183.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1184.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)

```



```

1185. );
1186. declare @assertion19 xml = '
1187.     <Assertion PositorUID="' + @BellaUID19 + '" Reliability="1.00" AssertionTime="2018-12-13 15:55">
1188.         <Posit AppearanceTime="2011-11-11 11:11">
1189.             <DereferencingSet>
1190.                 <Appearance ThingUID="' + @DisserUID19 + '" Role="husband" />
1191.                 <Appearance ThingUID="' + @BellaUID19 + '" Role="wife" />
1192.             </DereferencingSet>
1193.             <Value>married</Value>
1194.         </Posit>
1195.     </Assertion>
1196. ';
1197. exec Assert @assertion19 OUTPUT;
1198.
1199. -- now Bella has asserted her second marriage, but her second marriage took place without a church
1200. select * from [v_Assertion] where [Value] in ('married', 'divorced')
1201. order by [DereferencingSetUID], [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1202.
1203. -- another way to find this is from finding every assertion in which Bella has appeared in the wife role
1204. -- regardless if the relationship involves a church or not
1205. declare @BellaUID_Q1 varchar(555) = (
1206.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1207.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
1208. );
1209. select * from [v_Assertion]
1210. where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q1") and @Role = "wife"]') = 1
1211. order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1212. -- the above can be achieved through joins in the underlying model as well
1213.
1214. -- but, Bella made a mistake, the appearance time of her second marriage is not correct
1215. -- so she first makes a retraction of the erroneous assertion
1216. declare @DisserUID20 varchar(555) = (
1217.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1218.     cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[@Role = "has name"]') as T(Appearance)
1219. );
1220. declare @BellaUID20 varchar(555) = (
1221.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion

```

```

1222.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[Role = "has name"]') as T(Appearance)
1223. );
1224. declare @assertion20 xml = '
1225.     <Assertion PositorUID="' + @BellaUID20 + '" Reliability="0" AssertionTime="2018-12-13 16:00">
1226.         <Posit AppearanceTime="2011-11-11 11:11">
1227.             <DereferencingSet>
1228.                 <Appearance ThingUID="' + @DisserUID20 + '" Role="husband" />
1229.                 <Appearance ThingUID="' + @BellaUID20 + '" Role="wife" />
1230.             </DereferencingSet>
1231.             <Value>married</Value>
1232.         </Posit>
1233.     </Assertion>
1234. ';
1235. exec Assert @assertion20 OUTPUT;
1236.
1237. -- in the view of all assertions the retraction is now visible
1238. declare @BellaUID_Q2 varchar(555) = (
1239.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1240.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[Role = "has name"]') as T(Appearance)
1241. );
1242. select * from [v_Assertion]
1243. where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q2") and @Role = "wife"]') = 1
1244. order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1245.
1246. -- which means that the latest information we have is that Bella is divorced
1247. select * from [Information_in_Effect](getdate(), getdate())
1248. where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q2") and @Role = "wife"]') = 1
1249. order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1250.
1251. -- so Bella needs to assert a different posit, with the correct appearance time
1252. declare @DisserUID21 varchar(555) = (
1253.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1254.     cross apply AssertionXML.nodes('//*[Value = "Disser"]//*[Role = "has name"]') as T(Appearance)
1255. );
1256. declare @BellaUID21 varchar(555) = (
1257.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1258.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[Role = "has name"]') as T(Appearance)

```

```

1259. );
1260. declare @assertion21 xml = '
1261.     <Assertion PositorUID="' + @BellaUID21 + '" Reliability="1.00" AssertionTime="2018-12-13 16:00">
1262.         <Posit AppearanceTime="2012-12-12 12:12">
1263.             <DereferencingSet>
1264.                 <Appearance ThingUID="' + @DisserUID21 + '" Role="husband" />
1265.                 <Appearance ThingUID="' + @BellaUID21 + '" Role="wife" />
1266.             </DereferencingSet>
1267.             <Value>married</Value>
1268.         </Posit>
1269.     </Assertion>
1270. ';
1271. exec Assert @assertion21 OUTPUT;
1272.
1273. -- in the view of all assertions the correction is now visible
1274. declare @BellaUID_Q3 varchar(555) = (
1275.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1276.     cross apply AssertionXML.nodes('/*[Value = "Bella"]/*[@Role = "has name"']') as T(Appearance)
1277. );
1278. select * from [v_Assertion]
1279. where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q3") and @Role = "wife"']') = 1
1280. order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1281.
1282. -- new the latest information we have is that Bella is married, but note that the earlier divorce is also shown
1283. -- due to it having a different dereferencing set
1284. select * from [Information_in_Effect](getdate(), getdate())
1285. where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@ThingUID = sql:variable("@BellaUID_Q3") and @Role = "wife"']') = 1
1286. order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1287. -- there is currently no way for the system to know that these two relationships (with and without a church) both
1288. -- represent a marriage
1289.
1290.
1291. ----- MODELING -----
1292. /*
1293. ----- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----
1294. def. of a classifier and a class
1295. Let "is class" be a role reserved for the purpose of modeling.

```

```

1296. A posit pc = [{(C, is class)}, c, t], defines the name
1297. of a class through the string c and associates the unique
1298. identifier C with it. A classifier is a relationship that binds
1299. a thing to a class, expressed through posits on the form
1300. pM = [{(i, thing),(C, class)}, v, t].
1301. -----
1302.
1303. It is time for the Modeler to step in and tell us what some of our things are.
1304. */
1305.
1306. -- first create three classes: Person, Cathedral, and Church
1307. declare @ModelerUID22 varchar(555) = (
1308.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1309.     cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[Role = "has name"]') as T(Appearance)
1310. );
1311. declare @assertion22A xml = '
1312.     <Assertion PositorUID="' + @ModelerUID22 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1313.         <Posit AppearanceTime="1998-12-31 23:59">
1314.             <DereferencingSet>
1315.                 <Appearance ThingUID="C" Role="is class" />
1316.             </DereferencingSet>
1317.             <Value>Person</Value>
1318.         </Posit>
1319.     </Assertion>
1320. ';
1321. exec Assert @assertion22A OUTPUT;
1322. declare @assertion22B xml = '
1323.     <Assertion PositorUID="' + @ModelerUID22 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1324.         <Posit AppearanceTime="1998-12-31 23:59">
1325.             <DereferencingSet>
1326.                 <Appearance ThingUID="C" Role="is class" />
1327.             </DereferencingSet>
1328.             <Value>Cathedral</Value>
1329.         </Posit>
1330.     </Assertion>
1331. ';
1332. exec Assert @assertion22B OUTPUT;

```

```

1333. declare @assertion22C xml = '
1334.     <Assertion PositorUID=" + @ModelerUID22 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1335.         <Posit AppearanceTime="1998-12-31 23:59">
1336.             <DereferencingSet>
1337.                 <Appearance ThingUID="C" Role="is class" />
1338.             </DereferencingSet>
1339.             <Value>Church</Value>
1340.         </Posit>
1341.     </Assertion>
1342. ';
1343. exec Assert @assertion22C OUTPUT;
1344.
1345. -- list all classes
1346. select a.* from [v_Assertion] a
1347. where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') = 1
1348. order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1349.
1350. -- now associate some things with the classes
1351. declare @ModelerUID23 varchar(555) = (
1352.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1353.     cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
1354. );
1355. declare @ArchieUID23 varchar(555) = (
1356.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1357.     cross apply AssertionXML.nodes('//*[Value = "Archie"]//*[@Role = "has name"]') as T(Appearance)
1358. );
1359. declare @BellaUID23 varchar(555) = (
1360.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1361.     cross apply AssertionXML.nodes('//*[Value = "Bella"]//*[@Role = "has name"]') as T(Appearance)
1362. );
1363. declare @NDameUID23 varchar(555) = (
1364.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1365.     cross apply AssertionXML.nodes('//*[Value = "Notre-Dame de Paris"]//*[@Role = "has name"]') as T(Appearance)
1366. );
1367. declare @PersonUID23 varchar(555) = (
1368.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1369.     cross apply AssertionXML.nodes('//*[Value = "Person"]//*[@Role = "is class"]') as T(Appearance)

```

```

1370. );
1371. declare @CathedralUID23 varchar(555) = (
1372.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1373.     cross apply AssertionXML.nodes('//*[Value = "Cathedral"]//*[@Role = "is class"]') as T(Appearance)
1374. );
1375. declare @ChurchUID23 varchar(555) = (
1376.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1377.     cross apply AssertionXML.nodes('//*[Value = "Church"]//*[@Role = "is class"]') as T(Appearance)
1378. );
1379. declare @assertion23A xml = '
1380.     <Assertion PositorUID="' + @ModelerUID23 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1381.         <Posit AppearanceTime="1998-12-31 23:59">
1382.             <DereferencingSet>
1383.                 <Appearance ThingUID="' + @ArchieUID23 + '" Role="thing" />
1384.                 <Appearance ThingUID="' + @PersonUID23 + '" Role="class" />
1385.             </DereferencingSet>
1386.             <Value>active</Value>
1387.         </Posit>
1388.     </Assertion>
1389. ';
1390. exec Assert @assertion23A OUTPUT;
1391. declare @assertion23B xml = '
1392.     <Assertion PositorUID="' + @ModelerUID23 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1393.         <Posit AppearanceTime="1998-12-31 23:59">
1394.             <DereferencingSet>
1395.                 <Appearance ThingUID="' + @BellaUID23 + '" Role="thing" />
1396.                 <Appearance ThingUID="' + @PersonUID23 + '" Role="class" />
1397.             </DereferencingSet>
1398.             <Value>active</Value>
1399.         </Posit>
1400.     </Assertion>
1401. ';
1402. exec Assert @assertion23B OUTPUT;
1403. declare @assertion23C xml = '
1404.     <Assertion PositorUID="' + @ModelerUID23 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1405.         <Posit AppearanceTime="1998-12-31 23:59">
1406.             <DereferencingSet>

```

```

1407.         <Appearance ThingUID="' + @NDameUID23 + '" Role="thing" />
1408.         <Appearance ThingUID="' + @CathedralUID23 + '" Role="class" />
1409.     </DereferencingSet>
1410.     <Value>active</Value>
1411. </Posit>
1412. </Assertion>
1413. ';
1414. exec Assert @assertion23C OUTPUT;
1415. declare @assertion23D xml = '
1416.     <Assertion PositorUID="' + @ModelerUID23 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1417.         <Posit AppearanceTime="1998-12-31 23:59">
1418.             <DereferencingSet>
1419.                 <Appearance ThingUID="' + @NDameUID23 + '" Role="thing" />
1420.                 <Appearance ThingUID="' + @ChurchUID23 + '" Role="class" />
1421.             </DereferencingSet>
1422.             <Value>active</Value>
1423.         </Posit>
1424.     </Assertion>
1425. ';
1426. exec Assert @assertion23D OUTPUT;
1427.
1428. -- list all classes and classifiers
1429. -- as can be seen there are two things of class Person and one thing of class Cathedral and Church
1430. select
1431.     names.[Value] as [Name of Thing],
1432.     classes.[Value] as [Class of Thing]
1433. from (
1434.     select a.[Value],
1435.         Appearance.value('@ThingUID', 'uniqueidentifier') as ThingUID
1436.     from [v_Assertion] a
1437.     cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "has name"]') T(Appearance)
1438. ) names
1439. join (
1440.     select a.[Value],
1441.         DereferencingSet.value('Appearance[@Role = "thing"][1]/@ThingUID', 'uniqueidentifier') as ThingUID,
1442.         DereferencingSet.value('Appearance[@Role = "class"][1]/@ThingUID', 'uniqueidentifier') as ClassUID
1443.     from [v_Assertion] a

```

```

1444.     cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet[Appearance/@Role = "class" and Appearance/@Role = "thing"]') T
(DereferencingSet)
1445. ) classifiers
1446. on classifiers.ThingUID = names.ThingUID
1447. join (
1448.     select a.[Value],
1449.         Appearance.value('@ThingUID', 'uniqueidentifier') as ClassUID
1450.     from [v_Assertion] a
1451.     cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') T(Appearance)
1452. ) classes
1453. on classes.ClassUID = classifiers.ClassUID;
1454.
1455. -- the Modeler can clarify the situation of Notre Dame having two classes
1456. declare @ModelerUID24 varchar(555) = (
1457.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1458.     cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
1459. );
1460. declare @CathedralUID24 varchar(555) = (
1461.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1462.     cross apply AssertionXML.nodes('//*[Value = "Cathedral"]//*[@Role = "is class"]') as T(Appearance)
1463. );
1464. declare @ChurchUID24 varchar(555) = (
1465.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1466.     cross apply AssertionXML.nodes('//*[Value = "Church"]//*[@Role = "is class"]') as T(Appearance)
1467. );
1468. declare @assertion24 xml = '
1469.     <Assertion PositorUID="' + @ModelerUID24 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1470.         <Posit AppearanceTime="1998-12-31 23:59">
1471.             <DereferencingSet>
1472.                 <Appearance ThingUID="' + @CathedralUID24 + '" Role="subclass" />
1473.                 <Appearance ThingUID="' + @ChurchUID24 + '" Role="class" />
1474.             </DereferencingSet>
1475.             <Value>active</Value>
1476.         </Posit>
1477.     </Assertion>
1478. ';
1479. exec Assert @assertion24 OUTPUT;

```



```

1480.
1481. -- now Cathedral is a subclass of Church
1482. select
1483.     classes.[Value] as [Class],
1484.     subclasses.[Value] as [Subclass]
1485. from (
1486.     select a.[Value],
1487.           Appearance.value('@ThingUID', 'uniqueidentifier') as ClassUID
1488.     from [v_Assertion] a
1489.     cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') T(Appearance)
1490. ) classes
1491. join (
1492.     select a.[Value],
1493.           DereferencingSet.value('Appearance[@Role = "subclass"][1]/@ThingUID', 'uniqueidentifier') as SubclassUID,
1494.           DereferencingSet.value('Appearance[@Role = "class"][1]/@ThingUID', 'uniqueidentifier') as ClassUID
1495.     from [v_Assertion] a
1496.     cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet[Appearance/@Role = "class" and Appearance/@Role = "subclass"]') T
1497. ) inheritance
1498. on inheritance.ClassUID = classes.ClassUID
1499. join (
1500.     select a.[Value],
1501.           Appearance.value('@ThingUID', 'uniqueidentifier') as SubclassUID
1502.     from [v_Assertion] a
1503.     cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') T(Appearance)
1504. ) subclasses
1505. on subclasses.SubclassUID = inheritance.SubclassUID;
1506.
1507.
1508. /*
1509. ----- Excerpt from: Modeling Conflicting, Unreliable, and Varying Information -----
1510. def. of a posit type
1511. A posit type,  $\tau(p) = \{(C_1, r_1), \dots, (C_n, r_n)\}, \tau(v), \tau(t)$ , for a
1512. posit  $p = \{(i_1, r_1), \dots, (i_n, r_n)\}, v, t$ , is a structure constructed
1513. by replacing unique identifiers,  $ij$  with the unique identifiers of their
1514. class,  $C_j$ , the value,  $v$ , with its data type,  $\tau(v)$ , and the time point,  $t$ ,
1515. with its data type,  $\tau(t)$ .

```

```

1516. -----
1517.
1518. A posit type is a structure, but posits in the relational model can only have
1519. values of type varchar(max), so it cannot hold such a structure. We can, however,
1520. store it as serialized XML, where the XML has the same structure as the one we
1521. have been using to create posits.
1522. */
1523.
1524. -- let the Modeler create a posit type for the ternary "marriage" relationship
1525. declare @ModelerUID25 varchar(555) = (
1526.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1527.     cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
1528. );
1529. declare @PersonUID25 varchar(555) = (
1530.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1531.     cross apply AssertionXML.nodes('//*[Value = "Person"]//*[@Role = "is class"]') as T(Appearance)
1532. );
1533. declare @ChurchUID25 varchar(555) = (
1534.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1535.     cross apply AssertionXML.nodes('//*[Value = "Church"]//*[@Role = "is class"]') as T(Appearance)
1536. );
1537. declare @assertion25 xml = '
1538.     <Assertion PositorUID="' + @ModelerUID25 + '" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1539.         <Posit AppearanceTime="1998-12-31 23:59">
1540.             <DereferencingSet>
1541.                 <Appearance ThingUID="T" Role="is type" />
1542.             </DereferencingSet>
1543.             <Value>
1544.                 <Posit AppearanceTime="datetime">
1545.                     <DereferencingSet>
1546.                         <Appearance ThingUID="' + @PersonUID25 + '" Role="husband" />
1547.                         <Appearance ThingUID="' + @PersonUID25 + '" Role="wife" />
1548.                         <Appearance ThingUID="' + @ChurchUID25 + '" Role="church" />
1549.                     </DereferencingSet>
1550.                 <Value>varchar(555)</Value>
1551.             </Posit>
1552.         </Value>

```

```

1553.     </Posit>
1554. </Assertion>
1555. ';
1556. exec Assert @assertion25 OUTPUT;
1557.
1558. -- list the posit type
1559. select a.* from [v_Assertion] a
1560. where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is type"]') = 1
1561. order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1562.
1563. ----- TODO BELOW -----
1564. -- let the Modeler also add the posit type for the unary 'has beard' relationship
1565. declare @ModelerUID26 varchar(555) = (
1566.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1567.     cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[@Role = "has name"]') as T(Appearance)
1568. );
1569. declare @PersonUID26 varchar(555) = (
1570.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1571.     cross apply AssertionXML.nodes('//*[Value = "Person"]//*[@Role = "is class"]') as T(Appearance)
1572. );
1573. declare @assertion26 xml = '
1574.     <Assertion PositorUID="" + @ModelerUID26 + '' Reliability="1.00" AssertionTime="2018-12-01 00:00">
1575.         <Posit AppearanceTime="1998-12-31 23:59">
1576.             <DereferencingSet>
1577.                 <Appearance ThingUID="T" Role="is type" />
1578.             </DereferencingSet>
1579.             <Value>
1580.                 <Posit AppearanceTime="datetime">
1581.                     <DereferencingSet>
1582.                         <Appearance ThingUID="" + @PersonUID26 + '' Role="has beard" />
1583.                     </DereferencingSet>
1584.                 <Value>varchar(555)</Value>
1585.                 </Posit>
1586.             </Value>
1587.         </Posit>
1588.     </Assertion>
1589. ';

```

```

1590. exec Assert @assertion26 OUTPUT;
1591.
1592. -- list the posit types
1593. select a.* from [v_Assertion] a
1594. where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is type"]') = 1
1595. order by [PositorUID], [AppearanceTime] desc, [AssertionTime] desc;
1596.
1597.
1598. -- now the Modeler can express that a class appears in a certain posit type
1599. -- since this can be derived we can write some code that loops through posit types
1600. -- (provided that they all appear "commonly" for the classes)
1601. declare @ClassUID27 varchar(555);
1602. declare @PositTypeUID27 varchar(555);
1603.
1604. declare maps cursor static for (
1605.     select distinct
1606.         Appearance.value('@ThingUID', 'uniqueidentifier') as ClassUID,
1607.         pt.PositTypeUID
1608.     from (
1609.         select cast(a.Value as xml) as PositType,
1610.             AssertionXML.value('/Assertion[1]/Posit[1]/DereferencingSet[1]/Appearance[1]/@ThingUID', 'uniqueidentifier') as PositTypeUID
1611.         from [v_Assertion] a
1612.         where AssertionXML.exist('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is type"]') = 1
1613.     ) pt
1614.     cross apply PositType.nodes('/Posit/DereferencingSet/Appearance') as T(Appearance)
1615. );
1616.
1617. open maps;
1618. fetch next from maps into @ClassUID27, @PositTypeUID27;
1619.
1620. declare @ModelerUID27 varchar(555) = (
1621.     select Appearance.value('@ThingUID', 'varchar(555)') from v_Assertion
1622.     cross apply AssertionXML.nodes('//*[Value = "Modeler"]//*[Role = "has name"]') as T(Appearance)
1623. );
1624.
1625. while(@@FETCH_STATUS = 0)
1626. begin

```

```

1627. declare @assertion27 xml = '
1628.     <Assertion PositorUID="" + @ModelerUID27 + "" Reliability="1.00" AssertionTime="2018-12-01 00:00">
1629.         <Posit AppearanceTime="1998-12-31 23:59">
1630.             <DereferencingSet>
1631.                 <Appearance ThingUID="" + @ClassUID27 + "" Role="class" />
1632.                 <Appearance ThingUID="" + @PositTypeUID27 + "" Role="type" />
1633.             </DereferencingSet>
1634.             <Value>commonly</Value>
1635.         </Posit>
1636.     </Assertion>
1637. ';
1638. exec Assert @assertion27 OUTPUT;
1639.
1640. fetch next from maps into @ClassUID27, @PositTypeUID27;
1641. end
1642.
1643. close maps;
1644. deallocate maps;
1645.
1646. -- now the Church and Person classes are tied to the posit type for the ternary "marriage" relationship
1647. -- and Person should also be tied to the unary "has beard" relationship
1648. select
1649.     classes.[Value] as [Class],
1650.     classes.[ClassUID],
1651.     cast(posit_types.[Value] as xml) as [PositType]
1652. from (
1653.     select a.[Value],
1654.            Appearance.value('@ThingUID', 'uniqueidentifier') as ClassUID
1655.     from [v_Assertion] a
1656.     cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is class"]') T(Appearance)
1657. ) classes
1658. join (
1659.     select a.[Value],
1660.            DereferencingSet.value('Appearance[@Role = "type"][1]/@ThingUID', 'uniqueidentifier') as PositTypeUID,
1661.            DereferencingSet.value('Appearance[@Role = "class"][1]/@ThingUID', 'uniqueidentifier') as ClassUID
1662.     from [v_Assertion] a

```

```
1663.     cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet[Appearance/@Role = "class" and Appearance/@Role = "type"]') T
(DereferencingSet)
1664. ) typings
1665. on typings.ClassUID = classes.ClassUID
1666. join (
1667.     select a.[Value],
1668.         Appearance.value('@ThingUID', 'uniqueidentifier') as PositTypeUID
1669.     from [v_Assertion] a
1670.     cross apply AssertionXML.nodes('/Assertion/Posit/DereferencingSet/Appearance[@Role = "is type"]') T(Appearance)
1671. ) posit_types
1672. on posit_types.PositTypeUID = typings.PositTypeUID;
1673.
1674. -- 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
```