Modelling Data With Entities and Relationships

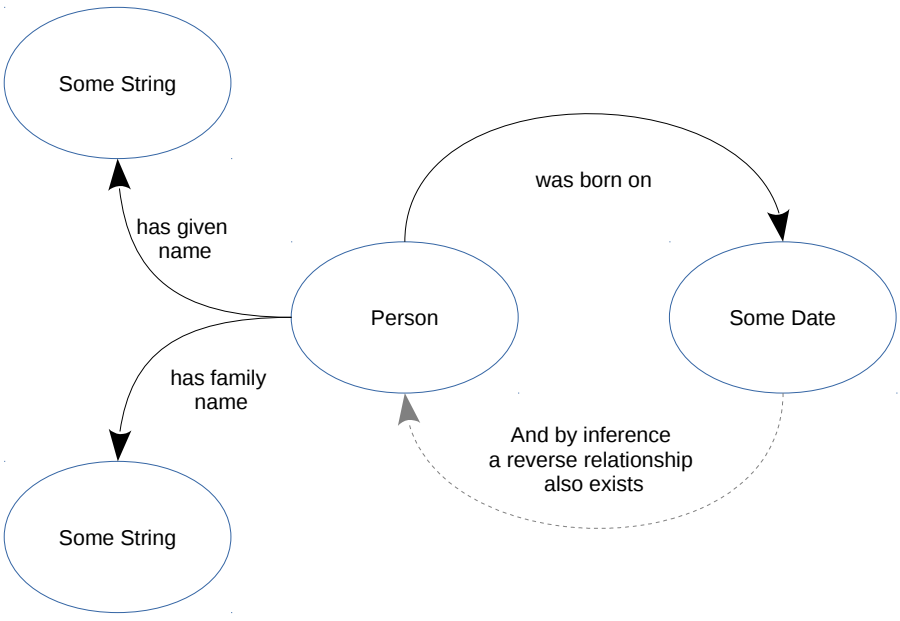
Overview

In an information technology context we often think of data as collections of attributes such as fields in a record. As an example here is a Person record.

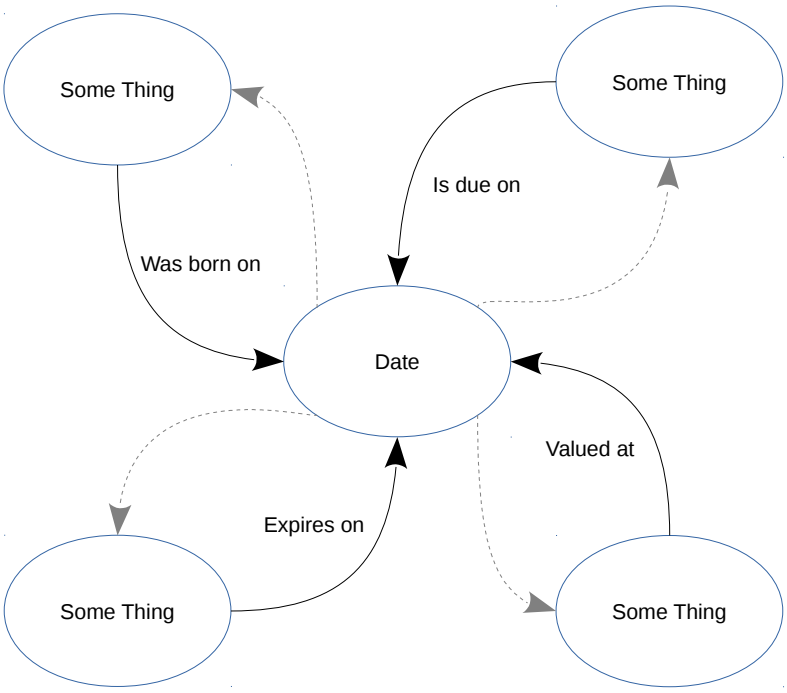
```
Person {
        String: given_name
        String: family_name
        Date: date_of_birth
        ...
}
```

Here we can see that each Person has, amongst other things, a date of birth. However even knowing the relevant portion of the storage as a date field its relevance is only resolved when we see its name from the Person layout. This means that to understand the data we need the schema and to change the schema means to alter the binary storage area for each instance of a Person record.

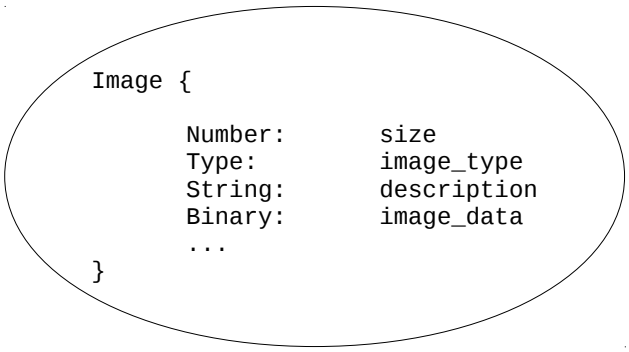
An alternative approach is to view any data entity as a composition of more primitive data types and the meaning of each primitive is conferred by a relationship from the entity to the component. Thus a Person entity might have a ‘was born on’ relationship to an entity that happens to be a Date.



Here the date is an abstract component and is only given relevance through the semantics of the relationship. This leads to an interesting consequence where the same date is used by many entities, each with its own relationship. The implied reverse relationship is then available to identify all entities which are related to this date which gives a comprehensive overview as to events relating to the date.

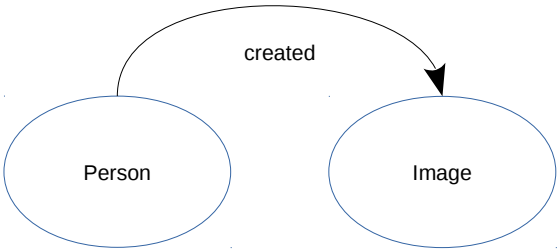


While restricting the number of date instances it is likely that this level of decomposition will have a negative impact on performance and resources and is therefore not preferable for large volumes of fixed format data. A more pragmatic approach is to associate a small number of relevant attributes to each entity and then to compose these into macro entities. Taking an image as an example

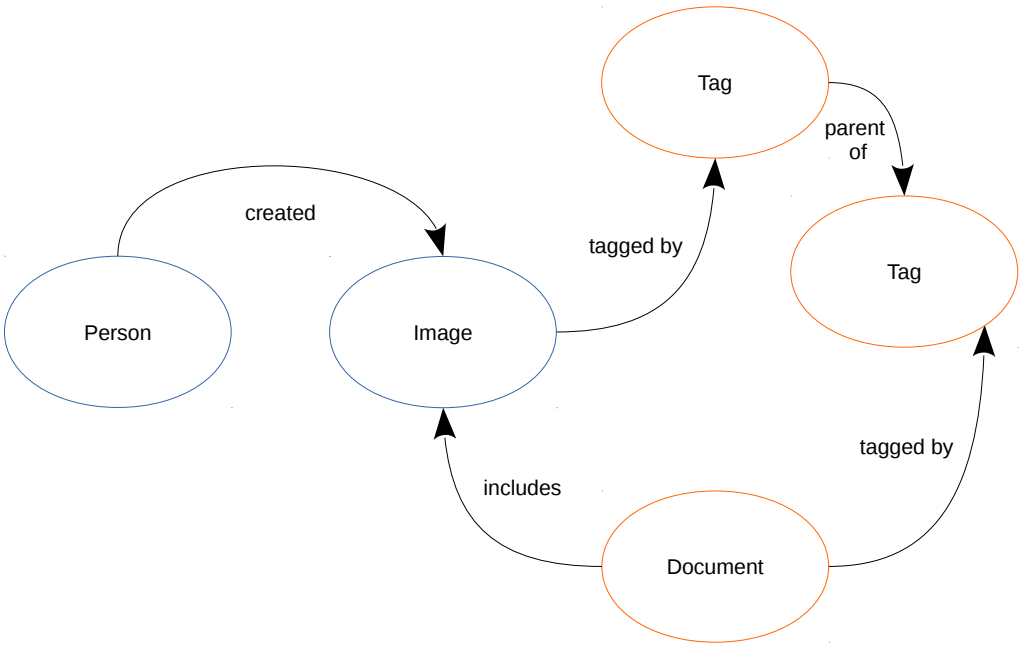


Here there is little to be gained from decomposing such things as name and size as the number of entities that share theses attributes will be very low.

This leads to a macro level compositional model. The significant advantage of this is that it is schema free where existing entities may be combined with newly created ones without code changes. Given a situation where we have Person and Image entities, an instance of a Person has a relationship to an Image they have created.



Subsequently the application is expanded to include new entities Document and Tag. These then may take part in relationships without altering the existing entities.

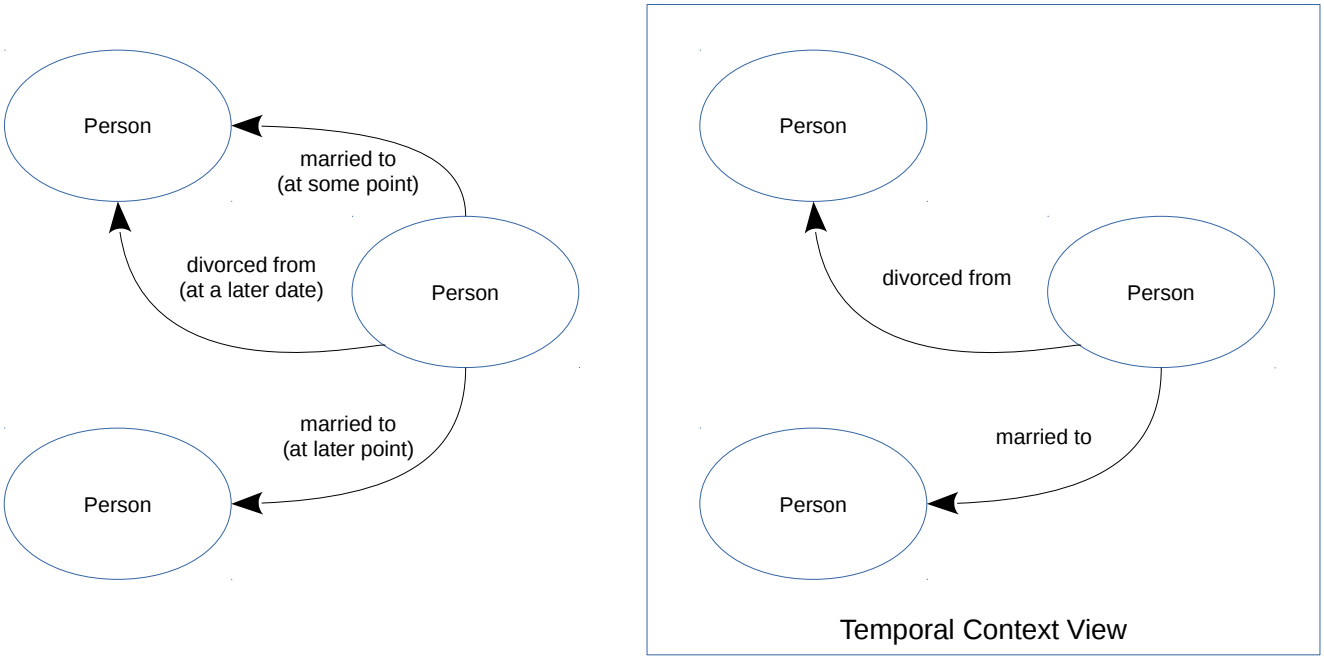


It is anticipated that a policy strategy for the composition of entities through relationships is desirable at an application level. Logically it may not make sense to assign Tag → *Relationship* → Document as opposed to the other way around.

Relationships

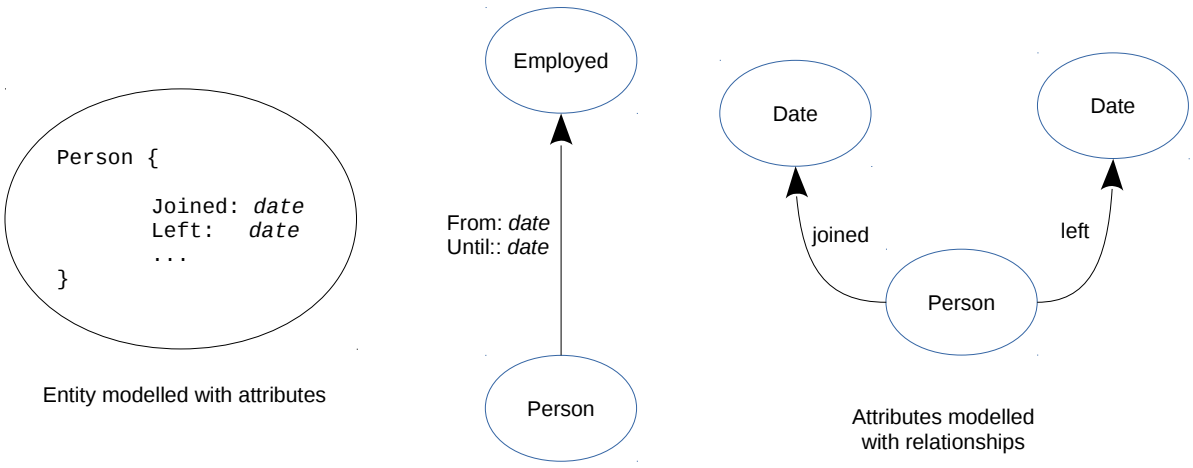
An important aspect of the compositional approach is that relationships can maintain state. Attributes that are not pertinent to an entity may become part of the relationship. An obvious example of this are *temporal* relationships, those that are only relevant relative to some point or points in time. Another example is to reduce visibility for some reason such as security where the relationship is only accessible to those performing a certain role.

In the following example it is assumed that being married is an exclusive state. The first construct is devoid of a temporal context, in the second a point in time after the divorce is assumed. Each relationship may hold different attributes, the ‘married to’ relationship might hold a date range and the ‘divorced from’ might simply hold a single event date.



Contexts may be composed using logic to further narrow the view.

Given that relationships have attributes there are a number of ways to model an entity. For instance an employee might be modelled in three different ways.



The first method suffers from a lack of extensibility, adding a second term employment might require altering the Person model or adding a duplicate Person node. The third example seems to place too much importance on the concept of date an an entity, such an approach may lead to an overwhelming number of relationships being attached to it. Furthermore it requires more edge traversals and adds complexity into ascertaining the employment status at any point in time. Finally it encourages the proliferation of relationship types.

The second approach seems to be the best compromise in that it can be achieved with an intuitive temporal type of relationship which lends itself to reuse, adding a second employment term is straightforward. Also further privilege constraints may be added to the ‘Employed’ state to restrict access, something that would be difficult to achieve in a ‘Date’ node given its widespread use.

Issues to consider with Relationships

A relationship has a direction ***FromEntity*** → ***Relationship*** → ***ToEntity*** as not all relationships are symmetrical. However a reverse of the relationship is also implied.

Cyclical paths may occur if two entities have relationships to each other. For instance Husband → *married to* → Wife and Wife → *married to* → Husband are both valid. Navigating a path from one to the other would need to be managed to prevent the process stalling.

As mentioned before a relationship Person → *was born on* → Date implies a reverse relationship. Navigating along reverse relationships is perfectly valid e.g. to find out all events for a given date but is also at risk of cyclical paths as by definition a relationship exists both ways between any two entities.

Deleting an entity implies that the relationships from that entity are also deleted. Depending on the nature of the relationship it may be desirable to implement constraints that alter the action of deletion.