Example 3.1.6.6. *Illustration of parent/child relationships between identifiers in related categories.*

```
loop_
  _struct_site.id
  _struct_site.details
    'P2 site C'
; residues with a contact < 3.7 Angstrom to an atom
  in the P2 moiety of the inhibitor in the
  conformation with _struct_asym.id = C
;
    'P2 site D'
; residues with a contact < 3.7 Angstrom to an atom
  in the P1 moiety of the inhibitor in the
  conformation with _struct_asym.id = D
;

loop_
  _struct_site_gen.id
  _struct_site_gen.site_id
  _struct_site_gen.label_comp_id
  _struct_site_gen.label_asym_id
  _struct_site_gen.label_seq_id
  _struct_site_gen.symmetry
  _struct_site_gen.details
    1  'P2 site C'  VAL  A   32  1_555  .
    2  'P2 site C'  ILE  A   47  1_555  .
    3  'P2 site C'  VAL  A   82  1_555  .
    4  'P2 site C'  ILE  A   84  1_555  .
    5  'P2 site D'  VAL  B  232  1_555  .
    6  'P2 site D'  ILE  B  247  1_555  .
    7  'P2 site D'  VAL  B  282  1_555  .
    8  'P2 site D'  ILE  B  284  1_555  .
```

Example 3.1.6.7. *A definition of an identifier which is parent to identifiers in other categories.*

```
save__struct_site.id
  _item_description.description
;     The value of _struct_site.id must uniquely
      identify a record in the STRUCT_SITE list.

      Note that this item need not be a number;
      it can be any unique identifier.
;
  loop_
    _item.name
    _item.category_id
    _item.mandatory_code
'_struct_site.id'             struct_site           yes
'_struct_site_gen.site_id'    struct_site_gen       yes
'_struct_site_keywords.site_id'
                              struct_site_keywords  yes
'_struct_site_view.site_id'   struct_site_view      yes
  loop_
    _item_linked.child_name
    _item_linked.parent_name
'_struct_site_gen.site_id'        '_struct_site.id'
'_struct_site_keywords.site_id'   '_struct_site.id'
'_struct_site_view.site_id'       '_struct_site.id'

  _item_type.code                 line
save_
```

DDL2 dictionaries each contain the definition for a single addressable concept.

For example, the three Miller index components of a diffraction reflection (`_diffrn_refln_index_h`, `_diffrn_refln_index_k`, `_diffrn_refln_index_l` that are described in the DDL1 core CIF dictionary in the data block `data_diffrn_refln_`) are described in a DDL2 dictionary in three separate save frames, `save__diffrn_refln.index_h`, `save__diffrn_refln.index_k` and `save__diffrn_refln.index_l`. In the DDL2 formalism, the intimate relationship between these three components is expressed through the common `_item_sub_category.id` value of `miller_index` and the mutual reference of the other Miller-index components by the `_item_dependent.dependent_name` entries in each separate save frame.

An apparent exception to this general rule is the case of save frames defining an item, often a category key, that is an identifier common to several categories. In this case, the save frame defining the 'parent' identifier implicitly defines the complete property set of each child identifier. For completeness, the respective child identifiers are each declared in their own save frames, but these act only as back references to the parent definition. This is explained more completely in Section 3.1.6.5.1 below.

3.1.6.5.1. *Inheritance of identifiers*

Example 3.1.6.6 is from an mmCIF of two related categories that describe characteristics of an active site in a macromolecular complex. The sites are described in general terms with a label and textual description in the STRUCT_SITE category (the first looped list in the example). Details of how each site is generated from a list of structural features form the STRUCT_SITE_GEN category (second loop or table).

It is clear that each instance of the data item `_struct_site_gen.site_id` in the second table must have one of the values listed as `_struct_site.id` in the first loop, because it is the purpose of these identifiers to relate the two sets of data: they are the glue between the two separate tables and must have the same values to ensure the referential integrity of the data set (that is, the consistency and completeness of cross-references between tables). Within a group of related categories like this, it is normal to consider one as the 'parent' and the others as 'children'.

Because all such linking data items must have compatible attributes, it is conventional in DDL2 dictionaries to define all the attributes in a single location, namely the save frame which hosts the definition of the 'parent' data item. In early drafts of DDL2 dictionaries, the 'children' were not referenced at all in separate save frames; software validating a data file against a dictionary was required to obtain all information about a child identifier from the contents of the save frame defining the parent. However, subsequent drafts introduced a minimal save frame for the children to accommodate dictionary browsers that depended on the existence of a separate definition block for each individual data item.

Consequently, the definition blocks in current DDL2 dictionaries conform to the structure in Example 3.1.6.7, which refers to the simple STRUCT_SITE example used above.

Note that the dependent data names are listed twice: once in the loop that declares their `_item.name` values and the categories with which they are associated; and again in a loop that makes the direction of the relationship explicit. A parent data item may have several children, but each child can have only a single parent (*i.e.* related data name whose value may be checked for referential integrity). Note also that each listed item has an `_item.mandatory_code` value of `yes`: because they are identifiers which link categories, they must be present in a table to allow the relationships between data items in different tables to be traced.

Other than the specific description text field, any declared attributes (in this example only the data type) have a common value across the set of related identifiers.

As mentioned above, it is not formally necessary to have a separate save frame for the individual children; but it is conventional to have such individual save frames containing minimal definitions that serve as back references to the primary information in the parent frame. These also provide somewhere for the specific text definitions for the children to be stored. The definition frame for `_struct_site_gen.id` is shown in Example 3.1.6.8.