Inter-Table Indexing in SNMP MIBs

David T. Perkins, 14-june-1998

This memo looks into indexing for SNMP tables when the indexing is used to describe relationships between tables. This issue is part of the larger issue, “How are relationships between tables in SNMP MIB modules shown?”

1.0 History

In the original use of SMIv1 in writing MIB modules, all relationships between tables are specified by text in the leading description section of a MIB document, in ASN.1 comments, or in the DESCRIPTION clauses for objects. Also all indices for a table are columns within the table.

As experience was gained writing MIB modules and a collection of MIB modules came to exist, it was noticed that many tables were simply the addition of columns to existing tables. In some cases, there was a *one-to-one dependent relationship* between rows in the first table and the second table, and in other cases there was a *sparse dependent relationship* between rows in the first table and the second table. However, in these sparse relationships, it is always the case that a row existed in the first table for each row in the second table. These two types of relationships between rows are examples of existence (or fate) relationships. In fate relationships, the deletion or creation of the base item results in the deletion and possible creation of the dependent item. There are other fate relationships besides these two.

Some MIB designers started specifying dependent-type relationships by using the indices of a base table in the INDEX clause of a dependent table. At about the same time that this change was occurring, SMIv2 was being developed. The issue of showing relationships between tables was discussed and addressed by the SMI through the addition of the AUGMENTS clause and clarifying text to describe when to use the AUGMENTS, when to use columns from a base table in the INDEX clause, and when to use columns within the table in the INDEX clause.

2.0 Existence Relationships

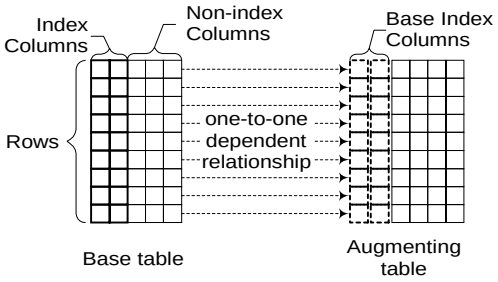
Objects typically have existence (or fate) relationships with several objects. The SMI uses the table model to group objects that share fate and are indexed identically. The relationships among objects (columns) in a table can be simple or complex. The relationship is simple if creation or deletion of any object in a row results in the same action for all the other objects in the row. Any other relationship is complex. Some of these complex relationships have been characterized by the EntryStatus and RowStatus textual conventions. Unfortunately, there is not a way that is parsable by a MIB compiler to specify the fate

relationships between the objects in a row. (The EntryStatus and RowStatus TCs provide only a framework. A person must read the descriptions of the objects to determine the relationships, and, thus, there may be misunderstandings between the MIB designer, agent and management station implementers, and network managers. To achieve inter-operation of SNMP agents and management applications there must be first achieved understanding of the MIB modules.)

Besides the fate sharing relationships between columnar objects in a table, there can be fate sharing relationships between tables. The SMI allows some of these relationships to be described with parsable constructs, but most relationships must be fully described with textual descriptions.

2.1 One-to-one Dependent Relationships

The SMI provides the AUGMENTS clause to specify a *one-to-one dependent relationship*. An example of the AUGMENTS clause is shown in Figure 1.



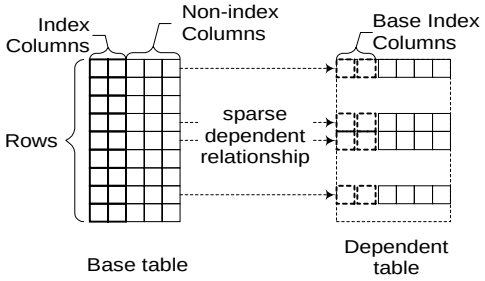

```
-- row definition in base table
baseEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { baseIndex1, baseIndex2 }
  ::= { baseTable 1 }

-- row definition in augmenting table
augEntry OBJECT-TYPE
  ... -- other clauses here
  AUGMENTS { baseEntry }
  ::= { augTable 1 }
```

Figure 1 One-to-one dependent relationship indicated with an AUGMENTS clause

The index columns in the augmenting table are “virtual.” They don't exist as columns in the augmenting table, but are conceptually part of the table for indexing. The augmenting table conceptually extends the base table with additional columns. Creation (or deletion) of a row in the base table results in the same fate for the row in the augmenting table.

2.2 Sparse Dependent Relationship

The SMI provides use of the INDEX clause containing all the indices of a base table to specify a *sparse dependent relationship*. An example is shown in Figure 2.



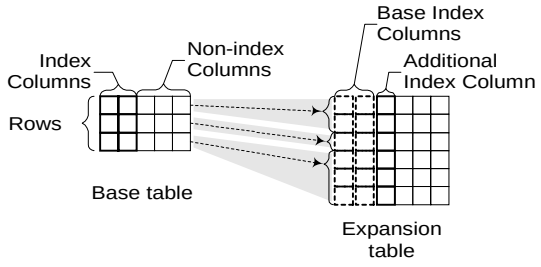

```
-- row definition in base table
baseEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { baseIndex1, baseIndex2 }
  ::= { baseTable 1 }

-- row definition in dependent table
dependEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { baseIndex1, baseIndex2 }
  ::= { dependTable 1 }
```

Figure 2 Sparse dependent relationship indicated with an INDEX clause

Just like an augmenting table, the index columns in the dependent table are “virtual.” They don't exist as columns in the dependent table, but are conceptually part of the table for indexing. The dependent table conceptually extends the base table with additional columns for those rows that have a specific attribute value. For example, if the base table contained information about network interfaces, a dependent table containing ethernet interface errors would containing only rows for those in the base table that were ethernet interfaces and not for serial or token ring interfaces. Creation (or deletion) of a row in the base table with the matching attribute value results in the same fate for the row in the dependent table.

2.3 Dependent Expansion Relationship

The only other type of relationship between two tables that can be specified by the index clause is a *dependent expansion relationship*. In it there is a base table on which a second table depends. The second table is indexed by columns from the base table and additional columns in the second table. An example is shown in Figure 3.




```
-- row definition in base table
baseEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { baseIndex1, baseIndex2 }
  ::= { baseTable 1 }

-- row definition in expansion table
expEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { baseIndex1, baseIndex2,
          expIndex1 }
  ::= { expTable 1 }
```

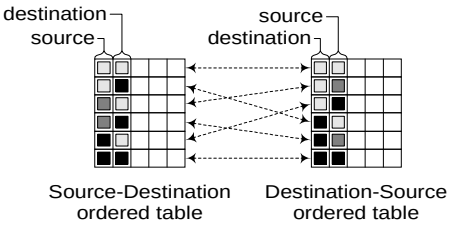
Figure 3 Dependent expansion relationship indicated with an INDEX clause

An expansion table is used to specify “an array within a table.” The base index columns in the expansion table are “virtual.” They don't exist as columns in the expansion table, but are conceptually part of the table for indexing. Additionally, the second table may only have rows if there are corresponding rows in the base table. There may be zero, one, or more rows in the expansion table for each row in the base table.

2.4 Reordering Relationship

There are other relationships between tables that can not be expressed through the INDEX clause. One common one is the result of providing the same information, but indexed differently. There are two predominant examples where two tables are specified to contain the same information. The first case is where the rows in the table are indexed by two columns, say, “i1” and “i2” and for efficiency the rows need to be retrieved by index order (i1, i2) and also by index order (i2, i1). The second case is where rows need to be retrieved in order by ranking and also by a distinguishing attribute value. In both of these cases, there

is *reordering relationship* between rows in the two tables. An example is shown in Figure 4.




```
-- row definition in source-dest table
sdEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { sdSrcIndex, sdDstIndex }
  ::= { sdTable 1 }

-- row definition in dest-source table
dsEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { dsDstIndex, dsSrcIndex }
  ::= { dsTable 1 }
```

Figure 4 Reordering relationship between tables

2.5 Multi-table Optimizations

Finally, there are “optimizations” that can be done when there are more than two tables. For example, Figure 5 shows a situation where a base table is both augmented and expanded.

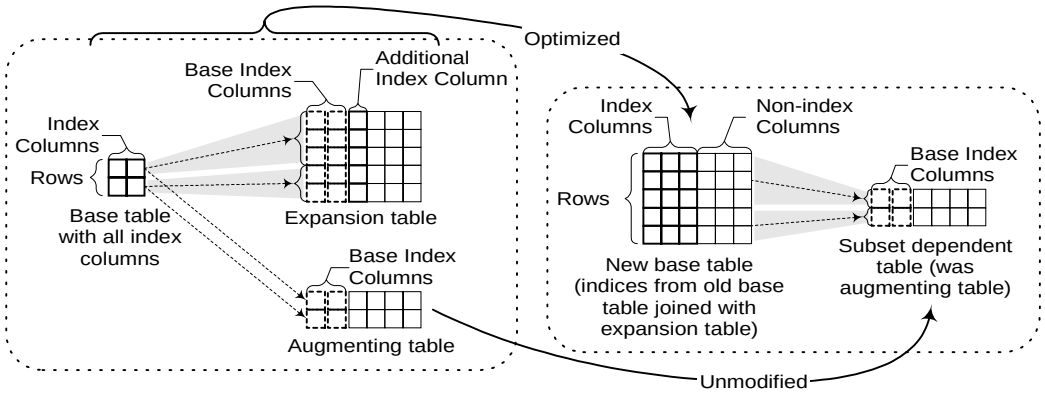
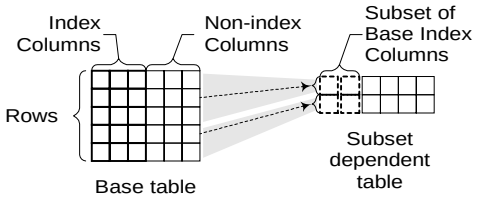


Figure 5 Optimization of relationships

However, the base table consists of only index columns, and, thus the MIB designer has chosen to combine the base table and the expansion table. The resulting base and subset dependent tables are shown in Figure 6.




```
-- row definition in base table
baseEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { baseIndex1, baseIndex2,
          baseIndex3 }
  ::= { baseTable 1 }

-- row definition in subset
--   dependent table
subEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { baseIndex1, baseIndex2}
  ::= { subTable 1 }
```

Figure 6 Subset dependent relationship

There are also other optimizations. Figure 7 shows the optimization of a base table and two expansion tables.

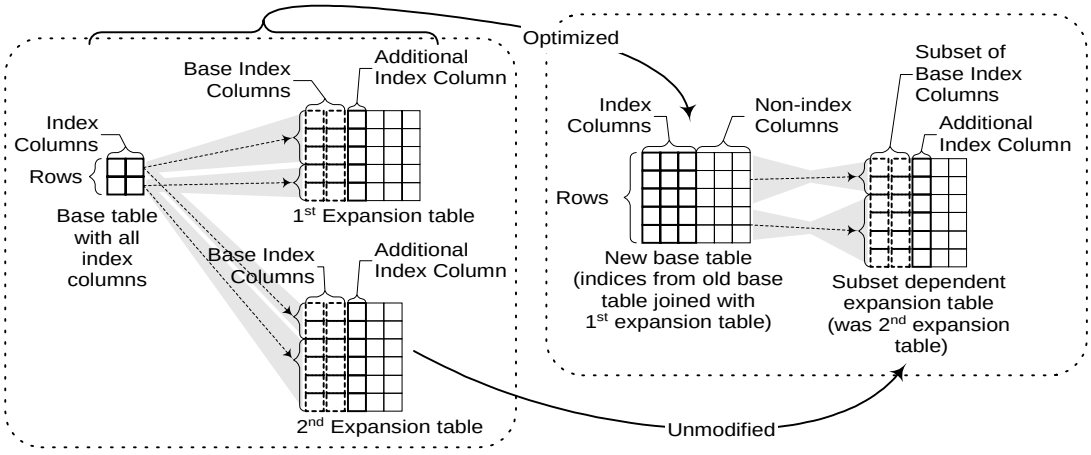
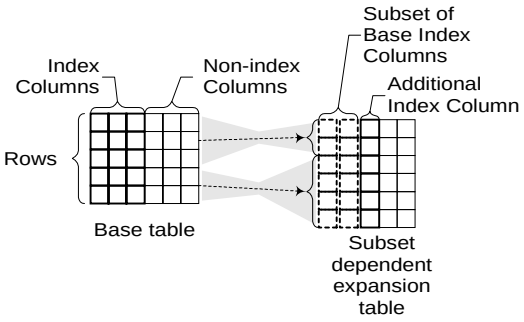


Figure 7 Optimization with two expansion tables

The MIB designer has chosen to combine the first expansion table with the base table. The resulting base and subset dependent expansion tables are shown in Figure 8. Rows in the subset dependent expansion table exist only if corresponding rows exist in the base table, and expand on them.




```
-- row definition in base table
baseEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { baseIndex1, baseIndex2,
          baseIndex3 }
  ::= { baseTable 1 }

-- row definition in subset
--   dependent expansion table
subEntry OBJECT-TYPE
  ... -- other clauses here
  INDEX { baseIndex1, baseIndex2,
          expIndex1 }
  ::= { subTable 1 }
```

Figure 8 Subset dependent expansion relationship

3.0 Rules from the SMI

Section 7.8.1 of the SMI contains the rules for determining how to create an INDEX or AUGMENTS clause. The text is:

> When defining instance identification information for a conceptual table:
>
> (1) If there is a one-to-one correspondence between the conceptual rows of this table and an existing table, then the AUGMENTS clause should be used.

> (2) Otherwise, if there is a sparse relationship between the conceptual rows of this table and an existing table, then an INDEX clause should be used which is identical to that in the existing table. For example, the relationship between RFC 1573's ifTable and a media-specific MIB which extends the ifTable for a specific media (e.g., the dot3Table in RFC 1650), is a sparse relationship.
>
> (3) Otherwise, if no existing objects have the required syntax and semantics, then auxiliary objects should be defined within the conceptual row for the new table, and those objects should be used within the INDEX clause for the conceptual row.

These rules clearly state all the cases for creation of an INDEX or an AUGMENTS clause. There has been no controversy in the interpretation of the first two rules. The third rule is the one where there have been differences of opinion as to its meaning. The first part of the controversy is the application of these rules. Some claim that this section is meant only to specify when to use an AUGMENTS clause instead of an INDEX clause. This claim is based on the section head, which is “Relation between INDEX and AUGMENTS clauses” and the section number. This claim is weak due to two factors. The first is the first sentence in the section. This sentence does not limit the section as claimed. The second factor is the third point in the rules is not needed if the section was only to determine when an AUGMENTS clause should be used. Once it is determined that this section is applicable, the second part of the controversy is the meaning of the phase “have the required syntax and semantics.” The sub-term “syntax” in the SMI means the data type of the index. The term “semantics” includes existence and indexing relationships. Thus, the chosen index objects must have the existence and indexing relationships that are compatible with those of the table. When you work out what is valid when the index items are from other tables, then the result is that only dependent expansion, subset dependent, and subset dependent expansion relationships can be legally specified.

4.0 Index Verification Procedure

Given that the existence and indexing relationships for index items must be compatible with a table, the following steps can be used to verity if the items in an INDEX clause are valid:

1. Check to see if any INDEX item is a scalar object. If so, then INDEX is invalid. (Scalar objects do not have compatible existence relationships with columnar objects.)
2. Check to see if any INDEX item is specified more than once in the INDEX list. If so, then the INDEX is invalid. (Duplicate items do not have compatible index relationships with the table.)
3. Check to see if all INDEX items are contained in the table. If so, then the INDEX list is valid.
4. Check to see that for all items in the INDEX clause, that if the item is contained in the table, then there is no item to its right in the INDEX clause that is a column in another

table. If so, then the INDEX is invalid. (This a reordering type relationship and the item does not have a compatible index relationship with the table.)

5. Moving from right to left in the INDEX clause, find the first item that is not a column in the table. If the INDEX clause of the table containing the item has a prefix that is identical to the prefix of the current INDEX clause, then the current INDEX clause is valid. If not, then the INDEX clause is invalid. For example, table t1Table has index clause "INDEX { t1C1, t1C2, t1C3, t1C4 }". If table t2Table has index clause "INDEX { t1C1, t1C2, t2C1 }", then it would be compatible. If table t2Table had index clause "INDEX { t1C1, t1C3, t2C1 }", then it would not be compatible, since index item t1C2 is not present. Note that clause "INDEX { t1C1, t1C2, t1C3, t2C1 }" would be valid, and index clause "INDEX { t1C1, t2C1 }" would also be valid. (Note that how the INDEX clauses are different determine the reason for the failure, which could be due to a reordering relationship, an incompatible index relationship, or an incompatible existence relationship.) Note, this step does not allow a nonindex columnar item specified in table t1 to be used as an index for another table.

(NOTE: A prefix of an INDEX clause is the ordered list of items from left to right in an INDEX clause up to and including an identified item. Such a list may contain zero, one, or more items.)

5.0 Summary

The INDEX and AUGMENTS clauses are used for two purposes. The first is to specify the indexing semantics, and the second is to show existence relationships with another table. Unfortunately, the SMI does not have the richness to specify in a parsable form all the common relationships between tables. These must be specified in textual comments. An attempt by MIB designers to use the INDEX clause to show some types of relationships has served only to muddle the relationships that may be specified according to the SMI. The result is added confusion in understanding a MIB module instead of clarity.