A Conceptual Model for Tables*

Xinxin Wang\(^1\) and Derick Wood\(^2\)

\(^1\) AT&T Labs, 200 Laurel Avenue, Middletown, NJ 07748, USA.
\(\text{xinxin@puma.att.com}\).

\(^2\) Department of Computer Science, Hong Kong University of Science & Technology,
Clear Water Bay, Kowloon, Hong Kong.
\(\text{dwood@cs.ust.hk}\).
WWW: \(\text{http://www.cs.ust.hk/~dwood}\).

Abstract. We describe a new, simple conceptual model for tables. The conceptual model treats a table as a map that has a domain which is a product of categories or index sets and a codomain which is a set of entry values. We demonstrate how we can use the model to specify the semantics of some tabular editing operations.

1 Introduction

Tables have been and are primarily a presentational technique, but with the surge of use of the Internet and computers, we expect that tables will be used even more. Moreover, they will be used in new ways. For example, tables are already being produced that are much too large to be displayed on a single page or in a single window. In this scenario, we need to have a sound conceptual model of tables to provide a foundation for the design of tabular browsers, the production of tabular views, and the design of tabular query systems.

The ideas we discuss are presented in a more rudimentary form in Wang’s thesis [21] and in even more rudimentary form in an early paper by Wang and Wood [22]. Vancirbeek [19, 20] appears to have been the first researcher to identify the multidimensional nature of tables and the hierarchical structure of rubrics, which we call categories or index sets depending on the context. Tables present a challenging problem for structured-document modelers as they do not fit comfortably in a hierarchical model as Furuta [10] observed in his thesis. Models that are more presentation oriented have been discussed by others [2, 3, 5, 7, 16, 17, 15] as well by the SGML community [12, 13, 14]. We could, of course, use a more complex apparatus for modeling tables, such as type theory or algebraic specifications, but our stance and philosophy is to develop small models using simple notions, rather than base models on more complex notions. This approach is similar to the “small languages” approach of Jon Bentley, Brian

* This paper will appear in the Principles of Digital Document Processing (PODDP 98) to be published in the Springer-Verlag Lecture Notes in Computer Science series. This work was supported under a grant from the Research Grants Council of Hong Kong.
Kernighan and others, and to the philosophy of system and program design. Books explores what may and will happen when systems are large and complex.

The content of a table is a collection of interrelated items that may be numbers, text, symbols, figures, mathematical equations, or even other tables. There are two kinds of items: the basic data displayed by a table, the entries, and the auxiliary data used to locate the entries, the labels.

We present a conceptual model for tables that is based on simple mathematical notions. It is independent of any specific display of a table, independent of any specific tool, and independent of any tabular management system. As an application of the model, we demonstrate how it can be used to specify the semantics of some simple editing operations.

We consider a table to be a region in a frame and before the borders there is a conceptual model's stock region, and the boxhead region contains the column headings, and the stub region contains the row headings. The stub is the lower left region that contains the index sets in the stub, and the boxhead is the upper right region that contains the column headings. The stub region is the upper right section that contains the column headings, the boxhead region is the lower left section that contains the row headings. A table is divided into four main regions by stub separation and boxhead separation. A table is divided into four main regions by stub separation and boxhead separation.

A conceptual model for tables that is based on simple mathematical notions. It is independent of any specific display of a table, independent of any specific tool, and independent of any tabular management system. As an application of the model, we demonstrate how it can be used to specify the semantics of some simple tabular editing operations.

We consider our tabular model to be the first step in addressing the problem: "What is a table?" It bears a similar relationship to tables that a context-free grammar model has to programming languages; there is much left.

### Table 1. The average marks for 1991-1992

<table>
<thead>
<tr>
<th></th>
<th>Assignments</th>
<th>Examinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winter</td>
<td>85</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>88</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Spring</td>
<td>75</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>Fall</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>75</td>
</tr>
</tbody>
</table>

We offer the three terms of 1991 and 1992, and the marks are the averages for each term. The marks for assignments and examinations are presented as average marks for the assignments and examinations of a course. The average marks are used to locate the entries. The marks for assignments and examinations are presented as average marks for the assignments and examinations of a course.
Some preliminary remarks

We present a conceptual model for tables that captures the underlying syntactic relationships of a table. The conceptual model treats a table as a map that has an unordered Cartesian product of categories or index sets as its domain and some universe of entries as its codomain. The index sets are restricted partial orders; indeed, they are trees. The number of index sets determines the dimension of the table and, as with programming language arrays, each entry in a d-dimensional table is determined by d indices. The method of indexing tables is what makes tables different from arrays and spreadsheets. We base the tabular model on a formalism for categories and index sets that is appropriate for tables.

Our position is that the manipulation of categories is the primary aspect of tabular manipulation whether it is for editing, querying, or formatting. In contrast, index sets for arrays and spreadsheets do not use and do not require such a rich repertoire of index-set operations.

One way we use a table such as Table 1 is that we have a specific year, term, and kind of mark in mind and want to retrieve the corresponding mark. For example, the mark corresponding to 1994, Spring, Examinations, and Final is 1997. We use the labels to index a unique entry. Observe that although the label Spring occurs twice in the table, we use only one of its appearances; namely, the one within 1994, Spring as the stub index.

Dewey decimal classification schemes, email addresses, and C++ structures use a similar notation. Similarly, Midterm and Final both depend on Examinations, so we write Examinations/Final as the box-head index.

Dependent labels define indices and sets of indices form index sets, which are hierarchical. Table 1 has six stub indices and six box-head indices, so the table has 36 entries (although some identical entries are presented only once). Conceptually, however, Table 1 may be viewed as a three-dimensional table since Year and Term may be treated as separate index sets. (Observe that we cannot break Marks into two or more index sets.) In this case, we need three indices to determine an entry uniquely just as we do with a three-dimensional array. Observe that there are two Year, three Term, and six Marks indices, so the conceptual table still has size 2 × 3 × 6 = 36.

As a reader, we use a Cartesian product, even when we do not think of it this way. For example, we may think of a product as a rectangle, even though the conceptual rectangle might be an area. The conceptual rectangle might be a table, even though there are two entries there. Indeed, we may think of a product as a table, even though there are two entries there. Each entry in a d-dimensional table is determined by d indices. The number of index sets determines the dimension of the table and, as with programming language arrays, each entry in a d-dimensional table is determined by d indices.

We present a conceptual model for tables that captures the underlying syntactic relationships of a table. The conceptual model treats a table as a map that has an unordered Cartesian product of categories or index sets as its domain and some universe of entries as its codomain. The conceptual model treats a table as a map that has an unordered Cartesian product of categories or index sets as its domain and some universe of entries as its codomain.
Examinations
Midterm Examinations
Final Grades

to specify the entries. There are six possible products, the one in Table 1 is the product
Year / Term / Marks
The other five different products are
Marks / Year / Term,
Term / Year / Marks,
Term / Marks / Year,
Marks / Term / Year.

An array is a random-access data structure for efficient storage and retrieval.

For tables and arrays (and spreadsheets), there are some similarities:
Both can be multidimensional and both are indexed by products of index sets.

But at this point the similarities end and we find only differences.

Product Order:
Each Cartesian product of the index sets determines a different array. For example, for the array
R[0][1][2]:

If we change the product order to
R[2][1][0]:
say, it is a different array.

For tables, each different product order determines the same table with a
different presentation.

Index Sets:
The ordering of the indices in an index set is total for arrays, but for tables it may be a total order, a partial order, or no
order at all. In addition, array index sets are not only a total order but
also a subrange of a total order (usually the set of integers). For
example, for the array
R[0][1][2]:

is a subrange of the integers, but the index
set
fWinter, Spring, Fallgf is not a subrange of the understood total order
fWinter, Spring, Summer, Fallgf. The index set for Marks is a partial order
but is not a total order.

Use:
Arrays are a data structure for the efficient storage and retrieval of similar
kinds of values. They provide random access to the stored values. Tables, on
the other hand, are a structure for the presentation and effective retrieval
of data; presentation is their primary use, whereas presentation is at most a
secondary use for arrays.

The manipulation of tables is, therefore, necessarily more complex than is the
manipulation of arrays. We want to be able to change the product of the
partition of the array. We need to be able to change the product of the
partition of the array. The different products correspond to the
same conceptual table presented in different ways. In addition, for each product of the
partition of the table, there are six possible products: the one in Table 1 is the

Examinations, Midterm Examinations, Final Grades,
We now return to the discussion of prefix-freeness. Although index sets such as \( A \subset \Sigma \) and the conceptual incoming edge of the root with \( z \), for some dotted string \( Y ; ; \), can be represented by a tree in which the root is labeled with \( z \); otherwise, it is the null dotted string \( \varepsilon \). For example, we can express this as a disjoint union of more than one index set. For example, we can also extend the discussion of prefix-freeness. Although this seems such that \( x = y \) for some dotted string \( Y ; ; \). The dotted string \( \varepsilon \) is in \( V \) if and only if \( x = \varepsilon \). Therefore, \( x \) is a prefix of \( y \) if and only if \( x = \varepsilon \). Clearly \( \varepsilon \) is always a prefix of \( x \). However, \( x \) is not a prefix of \( \varepsilon \). The dotted string \( \varepsilon \) is in the null set. We use strings and dotted strings to determine the prefix sets for indices. We have chosen \( \Sigma \) to be a finite set of symbols and \( \Sigma \) to be a function \( f \) from a finite collection \( I \) to be a prefix-free dotted-string set. We can capture the partitioning of a prefix-string over \( \varepsilon \). Denote by \( \alpha \), for some dotted string \( Y ; ; \), \( x \) is the null string over \( \varepsilon \). Then, a dotted string over the set \( \varepsilon \). We use strings and dotted strings to define the index sets for tables. We then choose \( \Sigma \). The conceptual model sets and maps to obtain the tabular model. We can extend the prefix-freeness concept to include the categorial labels directly in the dotted strings. We then use unordered Cartesian products of index sets and maps for the tabular model.
Given a prefix-free set \( X \) of dotted strings, we can partition it into prime subsets in a unique way. If the subsets are maximal, they are used to define prefix codes. A subset \( Y \) of \( X \) is a maximal prime set with respect to \( X \) if for all \( Y' \subseteq X \) such that \( Y' \) is a prefix of \( Y \), we obtain the following characterization of maximality:

**Proposition.** Let \( X \) be a prefix-free set. Then, \( X \) can be partitioned into a finite number of prime sets, maximal with respect to \( X \), and only if \( X \) is a maximal prime set with respect to \( X \).

In practice, we often use index sets that are not prime, for example, the index set Marks of Table 2 is one such example. However, we prefer to premultiply all dotted strings in the index set with a string to ensure that the index set is prime. For example, we can use Marks itself as such a string to obtain \( \text{Marks} \times \text{Assignments} \), and so on. This transformation is similar to that used in relational databases when we introduce a universal relation or in an object-oriented environment when we introduce a superclass of all classes. If we do not modify a nonprime index set in this way, then we may not obtain a partition into prime sets.

### Table 2: The average marks for 1991-1992

<table>
<thead>
<tr>
<th>Grade</th>
<th>Ass/1</th>
<th>Ass/2</th>
<th>Ass/3</th>
<th>Midterm</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winter</td>
<td>85</td>
<td>80</td>
<td>69</td>
<td>72</td>
<td>69</td>
</tr>
<tr>
<td>Summer</td>
<td>80</td>
<td>60</td>
<td>72</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>Fall</td>
<td>85</td>
<td>60</td>
<td>69</td>
<td>72</td>
<td>72</td>
</tr>
</tbody>
</table>

In practice, we often use index sets that are not prime. For example, Table 2 is one such example. However, we prefer to premultiply all dotted strings in the index set with a string to ensure that the index set is prime. For example, we can use Marks itself as such a string to obtain \( \text{Marks} \times \text{Assignments} \), and so on. This transformation is similar to that used in relational databases when we introduce a universal relation or in an object-oriented environment when we introduce a superclass of all classes. If we do not modify a nonprime index set in this way, then we may not obtain a partition into prime sets.
### Table 3. The average grades for 1991-1992.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

### Table 4. The average examination marks for 1991-1992.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Midterm</td>
<td>60</td>
<td>55</td>
<td>60</td>
<td>80</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Final</td>
<td>75</td>
<td>70</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
</tbody>
</table>

### Table 5. The average assignments marks for 1991-1992.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Winter 1</td>
<td>85</td>
<td>80</td>
<td>85</td>
<td>80</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Winter 2</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>Winter 3</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>
A set $X$ can have a prefix set and an index set. This is well-defined.

**Example:** Let $X = \{x, m \mid x \in X \land x = x \land m \in X \}$.

We leave the second operand to be an index set. Given a dotted-string set $X$, we denote the left quotient $X \div \theta$ of the dotted-string set $X$ and $\theta$.

**Theorem:** Let $X$ be a dotted-string set and $\theta$ be a dotted pre-fix of a dotted-string set $X$ and $\theta$. Then $X \div \theta$ is the set of all dotted strings $x$ in $X$ such that $x = x \land \theta$ for some dotted string $\theta$.

**Proof:**

We prove the notion of a dotted product of a dotted-string set and an index set. Let $X$ and $\theta$ be dotted-string sets and $\theta = x \land \theta$. We define $X \cdot \theta$ as the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted pre-fix of a dotted-string set $X$ and $\theta$. Let $X$ be a dotted-string set and $\theta$ be a dotted pre-fix of a dotted-string set $X$ and $\theta$. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is an index string.

We define the notion of a dotted pre-fix of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted pre-fix of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is an index string.

We define the notion of a dotted pre-fix of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted pre-fix of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is an index string.

We define the notion of a dotted pre-fix of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted pre-fix of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is an index string.

We define the notion of a dotted pre-fix of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted pre-fix of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is an index string.

We define the notion of a dotted pre-fix of an index set and an index set. Let $X$ and $\theta$ be index sets. Then $X \cdot \theta$ is the set of all index strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted product of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.

We define the notion of a dotted pre-fix of an index set and a dotted-string set. Let $X$ be a dotted-string set and $\theta$ be an index set. Then $X \cdot \theta$ is the set of all dotted strings $x$ in $X$ such that $x \cdot \theta$ is a dotted string.
We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more

defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more

defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more

defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more
defined. We define a more general notion of removal. However, there is more

we define

\[
\{\alpha \setminus \{x\}\} \cdot \mathcal{C} = \emptyset
\]

if and only if, there is no common prefix of the index strings in \(\{\alpha \setminus \{x\}\}\) and \(\mathcal{C}\). Clearly, we want to model this operation by a product of index sets; i.e.,

\[
\mathcal{C} \cap \{\alpha \setminus \{x\}\} = \emptyset
\]

We have used obvious abbreviations and \(\mathcal{O}\) is a new symbol. Note that

\[
\{\mathcal{J} \times \mathcal{D}, \mathcal{C}, \mathcal{D} \times \mathcal{J}, \mathcal{J} \times \mathcal{D}, \mathcal{J} \times \mathcal{J}, \mathcal{D} \times \mathcal{D}\} = \{\mathcal{J}, \mathcal{D}, \mathcal{C}\}
\]

where we have used obvious abbreviations and \(\mathcal{O}\) is a new symbol. Note that

\[
\{\mathcal{J} \times \mathcal{D}, \mathcal{C}, \mathcal{D} \times \mathcal{J}, \mathcal{J} \times \mathcal{D}, \mathcal{J} \times \mathcal{J}, \mathcal{D} \times \mathcal{D}\} = \{\mathcal{J}, \mathcal{D}, \mathcal{C}\}
\]

where we have used obvious abbreviations and \(\mathcal{O}\) is a new symbol. Note that

\[
\{\mathcal{J} \times \mathcal{D}, \mathcal{C}, \mathcal{D} \times \mathcal{J}, \mathcal{J} \times \mathcal{D}, \mathcal{J} \times \mathcal{J}, \mathcal{D} \times \mathcal{D}\} = \{\mathcal{J}, \mathcal{D}, \mathcal{C}\}
\]

where we have used obvious abbreviations and \(\mathcal{O}\) is a new symbol. Note that

\[
\{\mathcal{J} \times \mathcal{D}, \mathcal{C}, \mathcal{D} \times \mathcal{J}, \mathcal{J} \times \mathcal{D}, \mathcal{J} \times \mathcal{J}, \mathcal{D} \times \mathcal{D}\} = \{\mathcal{J}, \mathcal{D}, \mathcal{C}\}
\]

where we have used obvious abbreviations and \(\mathcal{O}\) is a new symbol. Note that
We can assume that categories $C$ and $D$ occur as the first two categories in $I$ and that $C$ occurs as the first category in $I$. Thus, $I$ is defined as:

$$I = (C \cdot D)$$

We have let $D$ be a number of issues introduced both implicitly or other.
Although this work is based on an earlier abstract model for tables, there is little difficulty in adapting it for the conceptual model we have described. All models have their upsides and downsides. On the upside, the conceptual model captures the hierarchical structure of index sets and their unordered combination—key properties in our opinion. The downside is that the frugality of the model moves the specification of orderings outside the model, although the user of a tabular system should be unaware of this fact. We chose this option since we wanted to separate structure from display; it is sufficient to label each set and domain with an integer place. Although these labels appear in the conceptual model we have described, little of this information is needed in our later models.
This article was processed using the TeX macro package with LNCs style.
Table 6. Correlation table — wheat and flour prices by months, 1941-1933.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

X = Wheat price per bushel in dollars
Y = Flour price per barrel in dollars

\[
X = Y^{1.23} + 4.56 \times 10^{-3}
\]