Over the past few years relational databases have become a standard field of study for researchers in computer science and this theory enjoys considerable mathematical support. In this chapter, we have attempted to give interesting results in a conceptually simple fashion and the main topics in the theory of relational databases have been discussed in detail with examples. The main body of the chapter is divided into five sections:

* Relational Model
* Data Dependencies
* Entropy and Information
* Lattices, Boolean Algebra, and Dependencies
* Normal Forms.

2.1 Relational Model

In this section, the basic definitions of relational model of databases are given. Most of the definitions are conventional ones but given in a simple form. Some
definitions are new. An attempt has been made to include all the fundamental ideas in relational database theory and examples are provided wherever necessary to illustrate the concepts.

A *datamodel*\(^{(13,30,38,56,60)}\) is primarily used for modelling a logical database in a DBMS. There are three data models proposed and available in the literature — the relational model\(^{(12,15,17-19)}\), the network model\(^{(14,58)}\), and the hierarchical model\(^{(59)}\). In this thesis we consider the relational model conceived by Codd in 1970. The relational model has many advantages over the other models mainly because of its simplicity and sound theoretical background. The model rests on the well developed mathematical theory of relations and the first-order logic\(^{(28,31)}\). In data modelling, the method stems from a perception of the real world through finite objects. The objects are essentially of three kinds: entities, their attributes, and the relationships among entities. Thus attributes describe entities which in turn are associated by relationships.

- An *entity* is a physical or abstract object that exists and can be distinguished from other objects.

For example, a Maruti 800 car with a registration number KGX 6018 is an entity since it uniquely identifies one particular car. By contrast, a book on databases published in 1982 is not an example of an entity since the features do not identify any particular book.

- Properties that are to represent an entity in the underlying database are called *attributes*.

For example, for an entity named *car* there may be two attributes: *Model* and *Registration Number* associated with it. It is important to distinguish between
attribute name and attribute value. For the attribute name model there may be attribute values like Maruti 800, Premier 118NE, Standard 2000, etc..

Before going into further definitions let us review the notations used in this chapter. For universal set of attributes, i.e., set of all attributes in a relation, we use $\Omega$. We use $U, V, W$ with or without subscripts for subsets of $\Omega$. $X, Y, Z$ with or without subscripts stand for variable attribute names and $A, B, C, \cdots$ with or without subscripts stand for specific attribute names. But when we discuss dependencies in terms of concepts in boolean algebra, small letters instead of capital letters are used for attribute names.

- The \textit{universe} of a relational database, denoted by $\Omega$, is a finite, nonempty set of attributes $A_1, \cdots, A_n$, i.e., $\Omega = \{A_1, \cdots, A_n\}$.

The \textit{domain} of an attribute $A_j$ written as $\text{DOM}(A_j)$, is a finite set of attribute values of $A_j$ which must be of the same datatype. A domain is a \textit{simple set} if all its elements are \textit{atomic} (i.e., nondecomposable by the underlying DBMS). Nonatomic attributes occur in the theory of nested relational databases\cite{35,52}. In relational databases only atomic attributes are considered. Domain of a subset $U$ of $\Omega$, denoted by $\text{DOM}(U)$, is the union of the domains of all attributes in $U$, i.e.,

$$\text{DOM}(U) = \bigcup_{A_k \in U} \text{DOM}(A_k)$$

where $\text{DOM}(\emptyset) = \emptyset$.

- A \textit{relation} $\mathcal{R}(\Omega)$ is a subset of a cartesian product $\times(D_1, \cdots, D_n)$ where $D_i = \text{DOM}(A_i)$.

Thus a relation can be represented by a function with domain $D_f = \times(D_1, \cdots, D_n)$ and range $R_f = \{0, 1\}$. Let that function be $P(d_1, \cdots, d_n)$ where $d_i \in D_i$. $P$ takes the value 1 if $(d_1, \cdots, d_n) \in \mathcal{R}$ and takes the value 0 otherwise. Speaking informally, a relation in relational database theory can be visualized as a table with each column
heading as attribute name and each row as an element of that relation. Under each attribute name there are attribute values of the same datatype. Cardinality of $\mathcal{R}$ or $\text{CARD}(\mathcal{R})$ is the number of rows in the table and degree of $\mathcal{R}$ or $\text{DEG}(\mathcal{R})$ is the number of columns. For example, a relation $\text{BOOK}$ may be defined as follows.

<table>
<thead>
<tr>
<th>Title</th>
<th>Author</th>
<th>ISBN</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Future Shock</td>
<td>Alvin Toffler</td>
<td>03199-11</td>
<td>Fiction</td>
</tr>
<tr>
<td>Bhagavad Gita</td>
<td>S. B. Varma</td>
<td>34-5691-2</td>
<td>Philosophy</td>
</tr>
<tr>
<td>One</td>
<td>Richard Bach</td>
<td>94770-08</td>
<td>Fiction</td>
</tr>
<tr>
<td>The $\TeX$book</td>
<td>D. E. Knuth</td>
<td>201-13448-9</td>
<td>Technical</td>
</tr>
<tr>
<td>A Suitable Boy</td>
<td>Vikram Seth</td>
<td>0-12-2318-7</td>
<td>Fiction</td>
</tr>
<tr>
<td>Power Shift</td>
<td>Alvin Toffler</td>
<td>04729-15</td>
<td>Fiction</td>
</tr>
<tr>
<td>Bhagavad Gita</td>
<td>S. Radhakrishnan</td>
<td>01-23-4568-9</td>
<td>Philosophy</td>
</tr>
</tbody>
</table>

A relation in table form has the following properties:

- There is no duplication of column names since these names are the attributes in the set $U$.
- There is no duplication of rows since $r_j$ is the set of tuples which occur in the domain of the function $P$.
- Row and column orders are insignificant. When columns are permuted for any reason, all the attribute values corresponding to it are also permuted.
- All attribute values are atomic.

- Let $U$ be a nonempty subset of $\Omega$, $\mathcal{R}(U)$ be a relation over $U$, and $V$ be a nonempty subset of $U$. The projection of $\mathcal{R}$ onto $V$ is the relation over $V$ consisting of the $V$-value of each tuple in $\mathcal{R}$, i.e.,

$\mathcal{R}[V] = \{ \mu[V] | V \subseteq U \text{ and } \mu \in \mathcal{R}(U) \}$.

In other words, a projection of a relation is obtained by deleting certain columns and by removing duplicate rows. In the above example $\text{BOOK}$ is a relation over...
$U = \{\text{Title, Author, ISBN, Type}\}$. The projection of $BOOK$ over $V = \{\text{Author, Type}\}$ will be:

<table>
<thead>
<tr>
<th>Author</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alvin Toffler</td>
<td>Fiction</td>
</tr>
<tr>
<td>S. B. Varma</td>
<td>Philosophy</td>
</tr>
<tr>
<td>Richard Bach</td>
<td>Fiction</td>
</tr>
<tr>
<td>D. E. Knuth</td>
<td>Technical</td>
</tr>
<tr>
<td>Vikram Seth</td>
<td>Fiction</td>
</tr>
<tr>
<td>S. Radhakrishan</td>
<td>Philosophy</td>
</tr>
</tbody>
</table>

Note that every projection is also a relation over a smaller set of attributes.

- Let $\mathcal{R}_1(U)$ and $\mathcal{R}_2(V)$ be two relations. The *natural join* (or simply join)[20] of $\mathcal{R}_1$ and $\mathcal{R}_2$ written as $*(\mathcal{R}_1, \mathcal{R}_2)$ is the relation with attributes from $U(V - (U \cap V))$ consisting of all tuples, such that for each tuple $r$ in the join there exists some tuple $s_1$ in $\mathcal{R}_1$ and $s_2$ in $\mathcal{R}_2$ satisfying $r[U] = s_1$ and $r[V] = s_2$.

When $U$ and $V$ are disjoint, a join of $\mathcal{R}_1$ and $\mathcal{R}_2$ will become the cross product of $\mathcal{R}_1$ and $\mathcal{R}_2$. As an example of join, consider the following relations $\mathcal{R}_1$ and $\mathcal{R}_2$:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>ab</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>aa</td>
<td>18</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>ac</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>20</td>
<td>pq</td>
</tr>
<tr>
<td>ac</td>
<td>25</td>
<td>pr</td>
</tr>
</tbody>
</table>

Then,

$$*(\mathcal{R}_1, \mathcal{R}_2) =$$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>ab</td>
<td>15</td>
<td>20</td>
<td>20</td>
<td>pq</td>
</tr>
<tr>
<td>8</td>
<td>ac</td>
<td>9</td>
<td>10</td>
<td>25</td>
<td>pr</td>
</tr>
</tbody>
</table>
A decomposition of a relation \( R \) into its projections \( R_1, \ldots, R_n \) is lossless if \( R \) can be reconstructed by joining these projections, i.e.,

\[
R = *(R_1, \ldots, R_n).
\]

For example, consider the relation \( R \) and its projections \( R_1, R_2, \) and \( R_3. \)

\[
\begin{array}{|c|c|c|}
\hline
R & A & B & C \\
\hline
a_1 & b_1 & c_1 \\
a_1 & b_2 & c_1 \\
a_1 & b_3 & c_2 \\
a_2 & b_1 & c_3 \\
\hline
\end{array}
\]

Here the decomposition of \( R \) into \( R_2 \) and \( R_3 \) is lossless since \( *(R_2, R_3) = R. \) But the decomposition \( R_1, R_3 \) is a lossy decomposition since \( *(R_1, R_3) \neq R. \)

In the next section, various data dependencies in a relation are discussed. Terms like keys, superkeys, etc. are defined and explained.

### 2.2 Data Dependencies

Data dependencies are constraints imposed on data in a database. In addition to a set of attributes, a set of data dependencies is also an essential part of a relation or database scheme. The class of functional dependencies was the first type of data dependencies to be studied and has since been thoroughly investigated by many researchers including Codd[17], who proposed the concept, Delobel and Casey[24], who developed a set of inference rules, and Armstrong[3], who gave a set of inference rules which are independent, sound, and complete. The relevance of functional dependencies in the design of a relational database scheme has been based on the observation that, in most cases, a set of functional dependencies plus
Databases and Lattices

one join dependency are enough to express the dependency structure of a relational database scheme. Also, they are essential for the assumption of the universal relation scheme[29,39]. Multivalued dependencies were independently introduced by Fagin[27], Zaniolo[63], and Delobel[23]. This class of dependencies is a generalization of the class of functional dependencies since multivalued dependencies provide (many-to-many) relations, where as functional dependencies provide only (many-to-one) functions. This section discusses data dependencies briefly. All the terms are defined and illustrated with examples.

2.2.1 Functional Dependencies

An attribute \( A \) functionally depends on another attribute \( B \) if, in the relation, whenever \( B \)-value becomes equal, \( A \)-value also becomes equal. More formally:

- Let \( \Omega \) be the universe and \( U \) be a subset of \( \Omega \). A **Functional Dependency**, abbreviated as FD, is a constraint on \( U \) and is of the form \( V \rightarrow W \) where, \( V \) and \( W \) are subsets of \( U \). Relation \( R(U) \) satisfies FD \( V \rightarrow W \), or \( V \rightarrow W \) holds in \( R(U) \), if for every two tuples in \( R(U) \), say \( r \) and \( s \), whenever their \( V \)-values are identical, their \( W \)-values are also the same, i.e.,

\[
(r[V] = s[V]) \Rightarrow (r[W] = s[W])
\]

In the relation \( BOOK \) defined earlier, we see that the attribute \( ISBN \) determines every other attribute, \( Author \) and \( Title \) together determines \( ISBN \), \( Author \) determines \( Type \), and \( Title \) determines \( Type \). The following relation

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_0 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_2 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_3 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>( c_3 )</td>
</tr>
</tbody>
</table>

has the dependencies \( AB \rightarrow C \) and \( C \rightarrow A \). It can be verified that whenever the \( A- \)
values and B-values coincide, the corresponding C-values also coincide. Similarly, whenever the C-values coincide, the corresponding A-values also coincide.

**Inference rules for FDs**

* **REFLEXIVITY:**
  
  \[(V \subseteq U) \Rightarrow (U \rightarrow V)\]

i.e., for attributes \(A, B,\) and \(C\) the FD \(ABC \rightarrow AB\) is always true. FD's of this kind are said to be trivial. Here \(ABC\) is an abbreviation for \(\{A, B, C\}\) and similarly \(AB\) for \(\{A, B\}\).

* **AUGMENTATION:**
  
  \[(U_3 \subseteq U_4 \text{ and } U_1 \rightarrow U_2) \Rightarrow (U_1U_4 \rightarrow U_2U_3)\]

Special cases: When \(U_3 = \phi, U_1U_4 \rightarrow U_2,\) When \(U_3 = U_4, U_1U_4 \rightarrow U_2U_4.\) When \(U_3 = U_4 = U_1, U_1U_1 \rightarrow U_1U_2\) or simply, \(U_1 \rightarrow U_1U_2.\)

* **TRANSITIVITY:**
  
  \[(U \rightarrow V \text{ and } V \rightarrow W) \Rightarrow (U \rightarrow W)\]

The following inference rules are deduced from the above three rules.

* **PSEUDO-TRANSITIVITY:**

  \[(U_1 \rightarrow U_2 \text{ and } U_2U_3 \rightarrow U_4) \Rightarrow (U_1U_3 \rightarrow U_4)\]

If \(U_3\) is replaced with \(\phi\) the transitivity rule is obtained. This rule is a generalisation of transitivity.

* **UNION or ADDITIVITY:**

  \[(U \rightarrow V \text{ and } U \rightarrow W) \Rightarrow (U \rightarrow VW)\]

* **DECOMPOSITION or PROJECTIVITY:**

  \[(U \rightarrow VW) \Rightarrow (U \rightarrow V \text{ and } U \rightarrow W)\]
Cardinalities of projections $U$ and $V$ completely determine whether $U$ functionally determines $V$ or not. The following result states this.

* Let $\text{CARD}([\mathcal{R}[\Omega]])$ or simply $\text{CARD}(\Omega)$ be the cardinality of the relation $\mathcal{R}$ over the set of attributes $\Omega$ as defined earlier. Let $U$ and $V$ be subsets of $\Omega$. Then $U$ is said to functionally determine $V$ if $\text{CARD}(U) = \text{CARD}(U, V)$.

* A set of attributes $U$ not containing any of the attributes in the attribute set $V$ is called a determinant of $V$ if $V$ is functionally dependent on $U$ and no proper subset of $U$ has this property. If $V$ is such that no super set of $V$ has $U$ as determinant then the set $V$ is called the colony of $U$. A set of attributes $U$ is called a saturated set if the following condition is satisfied: if a determinant is in $U$ then its colony must be in $U$.

The above definition is explained in the following example. Let $\mathcal{R}$ be the following relation.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_2$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_3$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_3$</td>
</tr>
</tbody>
</table>

There exist two FDs $AB \rightarrow C$ and $C \rightarrow A$ in $\mathcal{R}$. Determinants in $\mathcal{R}$ are $AB$ and $C$. Colony of $A = \phi$, $B = \phi$, $C = AC$, $AB = ABC$, $AC = AC$, $BC = ABC$, $ABC = ABC$. Saturated sets of $\mathcal{R}$ are $A$, $B$, $AC$, and $ABC$. Note that nullset $\phi$ and fullset $\Omega$ (set of all attributes) are saturated sets of every relation. Similarly, colony of $\phi = \phi$ and colony of $\Omega = \Omega$. We assume that no relation has a constant column.
Keys in a Relation

The central concept in the relational model is the concept of a key. Intuitively, an attribute (or a collection of attributes) may uniquely determine any particular tuple within a relation. The existence of at least one key is guaranteed by definition. However, the point is to have a key of minimal length, i.e., an attribute is to be discarded from the key provided it does not destroy its uniqueness.

- A set of attributes \( U \) is called a candidate key or simply a key of a relation, if \( U \) determines every attribute in the relation and no proper subset of \( U \) has this property. An FD \( K \rightarrow U \) is called a key dependency of \( R \) if \( K \) is a key of \( R \).

It is possible that a relation has more than one candidate key. In such cases, one of them is designated by the database designer as the primary key, i.e., as the primary means of identifying the corresponding tuples. All other keys are alternate keys. The choice of the primary key from among the candidate keys depends on the particular circumstances, though typically the shortest key is the most favourable. Any set containing a key is called a superkey.

- An attribute is called a prime attribute if it is an element of some key. Non-prime attributes are referred to as petty attributes. Two sets of attributes \( U \) and \( V \) are said to be equivalent if \( U \rightarrow V \) and \( V \rightarrow U \). A key is a simple key if it contains only one attribute. Complex keys are those in which there are more than one attribute. A set of attributes which determines at least one attribute outside it is called an open set. A set which is not an open set is called a closed set.

It can be verified that saturated sets and closed sets of a relation are identical. Now, consider the same example \( R \) with dependencies \( AB \rightarrow C \) and \( C \rightarrow A \).
Keys of the above relation are $AB$ and $BC$. $ABC$ is a super key since it contains a key. All the attributes are prime attributes. Since there are no keys with single attribute all the keys are complex. $C$, $AB$, and $BC$ are open sets since they determine $A$, $C$, and $A$ respectively. All other sets are closed.

The close connection between saturated sets (or closed sets) and lattices is given in section 2.4. Next we consider the representation of FDs in terms of boolean functions.

**Representation of FDs Using Binary Relations**

This section deals with a particular binary relation called an attribute relation and its specializations[45]. This binary relation is not to be confused with the relations of a relational database. The study of these relations can be helpful for visualizing FDs.

Let $\Omega$ be the set: $\Omega = \{X_{n-1}, X_{n-2}, \ldots, X_0\}$ where each $X_i$ is an attribute. Let $S$ be the power set of $\Omega$. Thus the cardinality of $S$ is $2^n$. Let the elements of $S$ be $R_0, R_1, \ldots, R_{2^n-1}$.

- An **attribute relation** (or **attribute graph**) is a subset of the Cartesian product $S \times S$. A **dependency relation** (or **dependency graph**) is an attribute graph satisfying the following three axioms.
  
  $\circ$ $T \cup U \rightarrow T$  (projectivity)
  
  $\circ$ $(T \rightarrow U$ and $T \rightarrow V) \Rightarrow (T \rightarrow U \cup V)$  (additivity)
\( (T \rightarrow U \text{ and } U \rightarrow V) \Rightarrow (T \rightarrow V) \) (transitivity)

where \( T, U, V \in S \) and the symbol \( \rightarrow \) stands for dependency relation.

It can be verified that the above axioms are independent. It is known [3,8,42] that these three axioms are appropriate for representing FDs and that they are sound and complete. Consider the attribute graph

\[ BA \rightarrow C, \quad C \rightarrow A \]

where the notation \( C, B, \) and \( A \) are used instead of \( X_2, X_1, X_0 \) and also \( BA \) instead of \( \{ B, A \} \) for convenience. The graph can be represented geometrically as in Fig. 2.1.

\begin{itemize}
  \item \( ABC \)
  \item \( AB \quad AC \quad BC \)
  \item \( A \leftarrow C \)
  \item \( B \)
  \item \( \phi \)
\end{itemize}

\[ \text{Fig. 2.1} \]

A graph can be represented by its adjacency matrix. The order of the adjacency matrix is equal to the number of vertices of the graph. In representing an attribute graph a ‘1’ appears at the intersection of row \( R_i \) and column \( R_j \) if \( R_i \rightarrow R_j \) otherwise a ‘0’ appears there. The adjacency matrix of the above graph is shown below.
Dependency closure of an attribute graph is defined as the minimal dependency graph which contains the given attribute graph.

The dependency closure of the above attribute graph is shown below.

\[
G = \begin{pmatrix}
\phi & A & B & AB & C & AC & BC & ABC \\
\phi & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
B & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
AB & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
C & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
AC & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
BC & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
ABC & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

The following results are derived in [45].

* The symmetric part of a dependency graph defines an equivalence relation.
The symmetric part of the graph in Fig. 2.1 is shown below.

\[
\begin{pmatrix}
\phi & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
B & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
BA & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
C & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
CA & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
CB & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
CBA & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1
\end{pmatrix}
\]

H \ = .

It is easy to verify that this matrix defines an equivalence relation. The equivalence classes are \( \phi, \{A\}, \{B\}, \{CA, C\}, \text{ and } \{BA, CB, CBA\} \).

* The symmetric part of a dependency graph defines a set of equivalence classes. In each class there is a unique maximal element, i.e., one element which contains all the others in the class.

The above result follows from the fact that the union of two nodes (attribute sets) in an equivalence class belongs to the same class and the maximal element is the union of all the elements in a class. The maximal elements in the above example are \( \phi, A, B, CA, \text{ and } ABC \).

* The maximal elements of the equivalence classes given by the symmetric part of the dependency graph are precisely the saturated sets.

A saturated set can be constructed as follows. Take any attribute set. If an attribute is found outside the set which is functionally determined by these, include that attribute in the set. Continue this process till the maximum possible attribute set is obtained. The resulting set will be a saturated set of that attribute relation.

Consider the attribute relation \( AB \rightarrow C \) and \( C \rightarrow A \). Consider the set \( \phi \).
Since it does not determine any attribute outside it, $\phi$ is a saturated set. Similarly, the sets $A$ and $B$ also form saturated sets of the relation. But the set $C$ can be expanded since it determines the attribute $A$ outside it. Since the set $CA$ does not determine any attribute, $CA$ forms another saturated set of the relation. Finally, the sets $AB$ and $ABC$ both expand to the same set and $ABC$ becomes a saturated set. It can be easily recognized that these sets are the very same maximal elements defined earlier.

* A saturated set functionally determines another saturated set if and only if the first one contains the second one.

Let $U$ and $V$ be two saturated sets. If $V \subseteq U$ then $U \rightarrow V$ follows from the definition of functional dependencies. Hence, the ‘only if’ part of the result follows directly. For the ‘if’ part, assume $U \rightarrow V$ and $V \not\subseteq U$. Let $Y$ be the attribute which is in $V$ but not in $U$. Since $U$ determines $V$, $U \rightarrow Y$ also is true. Since $U$ is a saturated set, by definition of saturated sets, $Y$ also should be in $U$.

• An attribute relation is called a *dissidence relation* if it satisfies the following axioms. The dissidence relation will be designated by the symbol $\not\rightarrow$. In the axioms that follow, $\Omega$ is the entire set of attributes, $\phi$ is the null set, $U$ and $V$ are subsets of $\Omega$ and $\bar{U}$ is the complement of $U$ with respect to $\Omega$.

  - $\Omega \not\rightarrow \phi$
  - $(U \not\rightarrow \phi) \Rightarrow (\bar{U} = V)$
  - $(U \not\rightarrow \bar{U} \text{ and } V \not\rightarrow \bar{V}) \Rightarrow (U \cap V \not\rightarrow \bar{U} \cap \bar{V})$

From the above axioms it can be seen that, if there is an arrow going out of a node, it has to go to the complement of that node. The symbol $\not\rightarrow$ physically means that the node on the left side does not functionally determine any of the attributes on the right side. Saturated sets defined earlier are the same as the sets on the left
side. From the saturated sets the boolean function representing the FDs can be obtained. This is shown in section 2.4.

More boolean relations like completion relation (defined later), equipotence relation and duals of all the above relations can be defined. The purpose of introducing these relations was to have a clear understanding of FDs through different perspectives. In the following section multivalued dependencies are described.

2.2.2 Multivalued Dependencies

Multivalued dependencies, abbreviated as MVDs, were introduced independently as a generalization of FDs by Fagin[27], Zaniolo[63], and Delobel[23]. In this section the properties of MVDs, their logical equivalence, some inference rules etc. are discussed along with necessary examples.

- A multivalued dependency (MVD) is statement of the form $U ightarrow V$, where $U$ and $V$ are subsets of $\Omega$. Let $W$ be the set of attributes not in $U$ or $V$, i.e., $W = \Omega - UV$. The MVD $U ightarrow V$ is said to hold for a relation if whenever there are tuples $s$ and $t$ in that relation where $s[U] = t[V]$, then there is a tuple $u$ where $u[UV] = s[UV]$ and $u[W] = t[W]$.

Consider the following relation having the MVD $BC \rightarrow D$.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>$c$</td>
<td>$a$</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>$c$</td>
<td>$f$</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>$e$</td>
<td>$a$</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>$e$</td>
<td>$f$</td>
</tr>
<tr>
<td>$d$</td>
<td>$a$</td>
<td>$c$</td>
<td>$a$</td>
</tr>
<tr>
<td>$d$</td>
<td>$a$</td>
<td>$c$</td>
<td>$f$</td>
</tr>
<tr>
<td>$d$</td>
<td>$a$</td>
<td>$e$</td>
<td>$a$</td>
</tr>
<tr>
<td>$d$</td>
<td>$a$</td>
<td>$e$</td>
<td>$f$</td>
</tr>
</tbody>
</table>
relation $R(\Omega)$ holds an MVD $U \rightarrow V$ if and only if $R(\Omega)$ holds $U \rightarrow W$ where $W = \Omega - (UV)$.

It can be seen that in the above example the MVD $BC \rightarrow A$ also exists. This can be proved with the help of the following formula.

$$U \rightarrow V = \bar{U} + V + W \quad \text{where} \quad W = \Omega - (UV)$$

Let $\Omega$ be the universe of the relation $R$. An MVD $U \rightarrow V$ holds in $R(\Omega)$ if and only if

$$R(\Omega) = * (R(UV), R(UW))$$

where $U, V \subseteq \Omega$, $W = \Omega - (UV)$, and $U \cap V = \Phi$.

Inference Rules for MVDs

As in the case of FDs, the following are some of the inference rules for MVDs.

* **Symmetry:**

$$U \rightarrow V \text{ iff } U \rightarrow W$$

where $UVW = \Omega$, and $W = \Omega - (UV)$.

* **Reflexivity:**

$$(V \subseteq U) \Rightarrow (U \rightarrow V)$$

* **Augmentation:**

$$(U_4 \subseteq U_3 \text{ and } U_1 \rightarrow U_2) \Rightarrow (U_1U_3 \rightarrow U_2U_4)$$

* **Transitivity:**

$$(U \rightarrow V \text{ and } V \rightarrow W) \Rightarrow (U \rightarrow W - V)$$
**PSEUDO-TRANSITIVITY:**

\[(U_1 \rightarrow U_2 \text{ and } U_2 U_3 \rightarrow U_4) \Rightarrow (U_1 U_3 \rightarrow U_4 - (U_2 U_3))\]

**UNION or ADDITIVITY:**

\[(U \rightarrow V \text{ and } U \rightarrow W) \Rightarrow (U \rightarrow VW)\]

**PROJECTIVITY:**

\[(U \rightarrow V \text{ and } U \rightarrow W) \Rightarrow (U \rightarrow V \cap W, \text{ } V \rightarrow V - W \text{ and } U \rightarrow W - V)\]

The first four inference rules are sound and complete for a set of MVDs on \(\Omega[8,54]\).

Other than the above inference rules, there are rules for both FDs and MVDs.

**REPLICATION:**

\[(U \rightarrow V) \Rightarrow (U \rightarrow V)\]

**COALESCENCE:**

\[(U_1 \rightarrow U_2 \text{ and } U_4 \rightarrow U_3, \text{ } U_3 \subseteq U_2, \text{ } U_2 \cap U_4 = \Phi) \Rightarrow (U_1 \rightarrow U_3)\]

\*(U \rightarrow V \text{ and } UV \rightarrow W) \Rightarrow (U \rightarrow W - V)\]

It has been shown that the above three inference rules are sound and complete[8].

These inference rules for FDs and MVDs written as logical formulas are also shown to be sound and complete[28,54].

Because of the equivalence theorem between data dependencies and a part of propositional logic, the determination of whether an FD \(U \rightarrow V\) or an MVD \(U \rightarrow V\) holds in a relation \(\mathcal{R}\) may be made by evaluating the corresponding formulas \(\bar{U} + V\) or \(\bar{U} + V + (\Omega - UV)\) for two tuples. Here \(\Omega\) is the universe of \(\mathcal{R}\).

In the next section one of the most important data dependencies and its properties are discussed. For any relation, at least one join dependency is necessary for the purpose of normalization.
2.2.3 Join Dependencies

These important data dependencies are based on the definition of join given earlier. In this section some of the results related to these dependencies and key dependencies are given.

- A **join dependency** (JD)\[50] is a statement of the form \*(U_1, U_2, \ldots, U_n)\ where, \(U_i \subseteq \Omega\). A relation \(\mathcal{R}\) is said to hold a JD \*(U_1, U_2, \ldots, U_n) if \(\mathcal{R} = \*(U_1, U_2, \ldots, U_n)\). Here \(U_1s\) are referred to as components of JD.

For example, \(\mathcal{R}(ABC)\) with the FDs \(C \rightarrow A\) and \(AB \rightarrow C\) has the JD \*(AB, BC). Note that the JD \*(U_1, U_2) is equivalent to the MVD \(U_1 \cap U_2 \rightarrow U_2\).

Let \(J_1\) and \(J_2\) be two JDs of \(\mathcal{R}\). If \(J_2\) is obtained by replacing two components of \(J_1\) by their union, then \(J_1\) logically implies \(J_2\). The membership algorithm\[26\] to find out whether a JD is a logical consequence of a set of KDs (key dependencies) is as follows:

Given a JD \*(U_1, U_2, \ldots, U_n)\ and a set \(\{K_1 \rightarrow \Omega, \ldots, K_s \rightarrow \Omega\}\) of KDs. Define a set \(J = \{U_1, \ldots, U_n\}\);

- Repeat
  - if \(K_i \subseteq V \cap W\) for some \(i, 1 \leq i \leq s\) and for some \(V, W \in J\)
  - then replace \(V\) and \(W\) in \(J\) by \(V \cup W\)
  - until no more steps are possible;

If any member of \(J\) is equal to \(U\) then the given JD is a logical consequence of the given set of KDs;

As in the case of FDs and MVDs there is a set of inference rules proposed for JDs too. Some of the other results will be discussed in later sections.

2.3 Entropy and Information

In this section we discuss the use of information theory\[36\] in the analysis of databases. For a better understanding of the theory, we have included many
results on the concept of entropy. Later we have shown how the entropy and data dependencies are connected. Further, it is shown that entropy can be used to define information in a relation. Every probability distribution has some uncertainty associated with it. The concept of entropy is to provide a quantitative measure of this uncertainty. Many axiomatic definitions of entropy are available in the literature [36] but the one available in [44] is particularly simple.

- The entropy of a random variable $X$ is given by

$$H(X) = -\sum_{k=1}^{n} p_k \log p_k$$

where $p_k$ is the probability of the variable taking the $k^{th}$ value.

For example, consider the following relation.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_2$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_3$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_3$</td>
</tr>
</tbody>
</table>

The various entropies with different variables are

- $H(A) = -\frac{3}{4} \log \frac{3}{4} - \frac{1}{4} \log \frac{1}{4} = 2 - \frac{3}{4} \log 3$
- $H(B) = -\frac{1}{2} \log \frac{1}{2} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} = \frac{3}{2}$
- $H(C) = -\frac{1}{2} \log \frac{1}{2} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} = \frac{3}{2}$
- $H(A + B) = -\frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} = 2$
- $H(A + C) = -\frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} = 2$
- $H(B + C) = -\frac{1}{2} \log \frac{1}{2} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} = \frac{3}{2}$
- $H(A + B + C) = -\frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{4} \log \frac{1}{4} = 2$
Now we will see how Venn diagrams can be used to represent entropies. The Venn diagram for a relation with three attributes is given in Fig. 2.2.

![Venn diagram for a relation with three attributes](image)

The entropy diagram with values of different areas for the previous example is given below in Fig. 2.3.

![Entropy diagram with values of different areas](image)

Here $H_2 = \frac{1}{2}$, $H_{13} = \frac{1}{2}$, $H_{23} = \frac{3}{4} \log 3 - \frac{1}{2}$, $H_{123} = \frac{3}{2} - \frac{3}{4} \log 3$. The entropies in different portions of the entropy diagram can be calculated with the help of the following formulas which follow directly from the *inclusion-exclusion principle*. 

1. \( H(A + B + C) = \\
\quad H(A) + H(B) + H(C) \\
\quad -H(AB) - H(AC) - H(BC) \\
\quad +H(ABC) \)

2. \( H(ABCDEF) = \\
\quad H(A + D + E + F) + H(B + D + E + F) + H(C + D + E + F) \\
\quad -H(A + B + D + E + F) - H(A + C + D + E + F) - H(B + C + D + E + F) \\
\quad +H(A + B + C + D + E + F) - H(D + E + F) \)

3. \( H(ABC\bar{D}\bar{E}\bar{F}) = \\
\quad H(ABC) - H(ABCD) - H(ABCE) - H(ABC\bar{F}) \\
\quad +H(\bar{A}BCDE) + H(ABCDF) + H(ABCEF) \\
\quad -H(ABC\bar{D}\bar{E}\bar{F}) \)

4. \( H(ABC) = \\
\quad H(A) + H(B) + H(C) \\
\quad -H(A + B) - H(A + C) - H(B + C) \\
\quad +H(A + B + C) \)

5. \( H(A + \bar{C}) = H(\bar{A}\bar{C}) + H(A) \)

Let \( H_k \) be the entropy of a set of variables written in terms of entropies of at least \( k \) variables. We find some interesting results that can be generalized into a simple formula. We will also see what \( H_n \) means in an entropy diagram of three variables. We will first consider a few examples and later generalize it.

1. For three variables we have the following results.

a) \( H_1(A + B + C) = \\
\quad H(A) + H(B) + H(C) \\
\quad -H(AB) - H(AC) - H(BC) \\
\quad +H(ABC) \)
b) \( H_2(A + B + C) = \)
\[
H(AB) + H(BC) + H(AC)
- 2H(ABC)
\]
c) \( H_3(A + B + C) = H(ABC) \).

2. For four variables we have

a) \( H_1(A + B + C + D) = \)
\[
H(A) + H(B) + H(C) + H(D)
- H(AB) - H(AC) - H(AD) - H(BC) - H(BD) - H(CD)
+ H(ABC) + H(ABD) + H(ACD) + H(BCD)
- H(ABCD)
\]
b) \( H_2(A + B + C + D) = \)
\[
H(AB) + H(AC) + H(AD) + H(BC) + H(BD) + H(CD)
- 2[H(ABC) + H(ABD) + H(ACD) + H(BCD)]
+ 3H(ABCD)
\]
c) \( H_3(A + B + C + D) = \)
\[
H(ABC) + H(ABD) + H(ACD) + H(BCD)
- 3H(ABCD)
\]
d) \( H_4(A + B + C + D) = H(ABCD) \)

3. Proceeding in similar way we find that

a) \( H_1(A + B + \cdots + N) = \)
\[
[\text{sum of entropies with one variable}]
- [\text{sum of entropies with two variables}]
+ [\text{sum of entropies with three variables}]
\,
\vdots
\]
\[
(-1)^{n-1} [\text{sum of entropies with all variables}]
\]
b) \( H_2(A + B + \cdots + N) = \)
[sum of entropies with two variables]
-2 [sum of entropies with three variables]
+3 [sum of entropies with four variables]

\[ (-1)^{n-1}(n - 1) \] [sum of entropies with all variables]

c) \( H_3(A + B + \cdots + N) = \)

[sum of entropies with three variables]
-3 [sum of entropies with four variables]
+6 [sum of entropies with five variables]

\[ (-1)^{n-2}\binom{n-1}{2} \] [sum of entropies with all variables]

a) \( H_4(A + B + \cdots + N) = \)

[sum of entropies with four variables]
-4 [sum of entropies with five variables]
+10 [sum of entropies with six variables]

\[ (-1)^{n-3}\binom{n-1}{3} \] [sum of entropies with all variables]

Continuing in the same fashion we have the general formula

\[ H_{m+1}(A + B + \cdots + N) = \]

\[ \binom{m}{m} \] [sum of entropies with \( m \) variables]
\[ -(\binom{m+1}{m}) \] [sum of entropies with \( m + 1 \) variables]
\[ +\binom{m+2}{m} \] [sum of entropies with \( m + 2 \) variables]

\[ (-1)^{n-m-1}\binom{n-1}{m} \] [sum of entropies with all variables]
2.3.1 Entropy and Dependencies

In this section we see how entropy can be used to obtain the data dependencies in a relation. First let us consider an example. Let the relation \( R(A, B, C) \) be

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_0 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_2 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_3 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>( c_3 )</td>
</tr>
</tbody>
</table>

Using the earlier calculated entropy values (see p. 24) we get

\[ H(AC') = 0 \quad \text{and} \quad H(\bar{A}BC) = 0 \]

The entropy diagram is given below in Fig. 2.4.

In the above diagram, entropies in the shaded regions \( \bar{A}BC \), \( AB\bar{C} \), and \( ABC \) are zero. We can directly obtain the dependency function \( f \).

\[
f = \bar{a}bc + ab\bar{c} + a\bar{b}c
\]

\[
= \bar{a}bc + \bar{c}a
\]
The entropy of each region shows how much dependency exists between those attributes. The more the dependency, the less the entropy. When the attributes become fully dependent the entropy becomes zero. In section 2.4 we have shown that a unique lattice can be constructed from the entropy diagram.

2.3.2 Information in a Relation

We have attempted to define information in a relation and in a database which is in Boyce–Codd normal form. Moreover, information in a lossless join is also discussed. The starting point of a model for database performance is the measure of the information stored in the database.

- Information, denoted as $I$ in a relation is the sum of entropies of all dependent attributes taken individually.

For example, let us calculate the information in the following relation.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_2$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_3$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_3$</td>
</tr>
</tbody>
</table>

The dependencies in the above relation are $AB \rightarrow C$, and $C \rightarrow A$. Entropies of attributes $A$ and $C$ are given below.

\[
H(A) = -\frac{3}{4} \log \frac{3}{4} - \frac{1}{4} \log \frac{1}{4} \\
= 2 - \frac{3}{4} \log 3
\]
Information in this relation $I = \text{Entropy of } A + \text{Entropy of } C$

\[
H(C) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - \frac{1}{4}\log\frac{1}{4}
\]

\[
= \frac{1}{2} + \frac{1}{2} + \frac{1}{2}
\]

\[
= \frac{3}{2}
\]

\[
I = H(A) + H(C)
\]

\[
= 2 - \frac{3}{4}\log 3 + \frac{3}{2}
\]

\[
= \frac{7}{2} - \frac{3}{4}\log 3
\]

The following features of information can be observed.

- The more the dependencies, the more the information.
- If there are no dependencies there is zero information.
- If fully dependent, it will have complete information.

In a relation which is in BCNF, we have at least one dependent attribute in each projection. We take marginal distribution corresponding to each attribute. Take the dependent attributes in the key. Thus,

- Information in a database, which is in BCNF, can be defined as the sum of informations in each relation (projections) in the decomposition.

Information in a database, i.e., the sum of informations in the projections should be equal to the information in the original relation. For example, consider the earlier relation:

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a_0$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td></td>
<td>$a_0$</td>
<td>$b_2$</td>
<td>$c_1$</td>
</tr>
<tr>
<td></td>
<td>$a_0$</td>
<td>$b_3$</td>
<td>$c_2$</td>
</tr>
<tr>
<td></td>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_3$</td>
</tr>
</tbody>
</table>

Here information $I = \frac{7}{2} - \frac{3}{4}\log 3$. The BCNF decomposition of the above relation is given below.
Information in the relation $\mathcal{R}(B, C)$, $I_1 = \frac{3}{2}$ and information in the relation $\mathcal{R}(A, C)$, $I_2 = 2 - \frac{3}{4}\log 3$. Thus total information in the database

$$I = I_1 + I_2$$

$$= \frac{7}{2} - \frac{3}{4}\log 3$$

Thus information in a relation is the same as information in its BCNF decomposition. From the above results we can have a new definition of lossless decomposition in terms of information.

- A set of projections of a relation is called *lossless decomposition* of the relation, if and only if the information in the join of projection is the same as the information in the original relation. Also, being in BCNF, the join of this decomposition gives back the original relation.

From properties of join, it can be seen that information in the projection can never be greater than the information in the original relation. *Inconsistent* projections are those projections whose join does not give the original relation back. Otherwise, they are *consistent*.

Now we will see how entropy diagrams can be used to find information in a relation. Consider the earlier relation with dependencies $AB \rightarrow C$ and $C \rightarrow A$. The corresponding entropy diagram will be
Due to dependency $AB \rightarrow C$, region corresponding to the set $ABC$ will have zero entropy. Similarly $C \rightarrow A$ will make the entropies of the regions $ABC$ and $A\overline{B}C$ zero. Let the entropies of the other regions be $e_1$, $e_2$, $e_3$, and $e_4$ as given in Fig. 2.5. Then the entropies of each dependent attribute can be obtained by adding the entropies of the regions which fall in the set corresponding to that attribute. Thus

$$\text{Information } I = \text{Entropy of } A + \text{Entropy of } B$$

$$= (e_1 + e_2) + (e_1 + e_2 + e_3)$$

$$= 2e_1 + 2e_2 + e_3$$
Consider another example of a relation having the dependencies \( A \rightarrow B \) and \( B \rightarrow C \). The corresponding entropy diagram is shown in Fig. 2.6.

Information \( I = \text{Entropy of } B + \text{Entropy of } C \)

\[
I = (e_1 + e_2) + (e_1) \\
= 2e_1 + e_2
\]

Even when we add the dependency \( A \rightarrow C \), we find that the entropy diagram remains unchanged and the information remains the same. This is due to the fact that \( A \rightarrow C \) is not a new dependency as it can be deduced from original dependencies by transitivity.

The next section explores the connection between lattices, boolean algebra, and data dependencies. All the important terms are defined and explained. Some of the definitions are altered from conventional ones to make understanding easier.

2.4 Lattices, Boolean Algebra, and Dependencies

This section deals with one of the most important structures in the theory of relational databases. It is shown that lattices\([9, 25]\) and dependencies in relations have a one-to-one correspondence and the study of these discrete structures can give deeper insight into the database theory. Similarly, boolean algebra also plays an important role here. In this section, the close connection between these structures is studied in detail. All the necessary definitions are provided with examples as the chapter proceeds.

2.4.1 Lattices and Dependencies

In this section, lattices are visualized in different ways by giving equivalent definitions. Known results of the matrix theory, data dependencies, etc. are used to arrive at the results. Before going into these details, some basic terms have to be defined.
• A digraph[46] is said to be nonsymmetric if edge

\[(x, y) \text{ and } (y, x) \Leftrightarrow (x = y)\]

i.e., there are no symmetric edges other than loops at every node. A nonsymmetric, transitive graph is called a partial order.

Reduction of a partial order can be obtained by removing the loops and transitive edges. A digraph which is a partial order is given in Fig. 2.7.a and its reduction is given in Fig. 2.7.b. All the edges are assumed to have downward arrows.

A partial order can be defined in a different way as follows.

• Let \( S \) be a set of elements \( a, b, c, \cdots \). Then a relation \( \mathcal{O} \) of partial order over \( S \) is any dyadic relation over \( S \) which is:

  i) reflexive: for every \( x \in S \), \( x \mathcal{O} x \);
  ii) antisymmetric: if \( x \mathcal{O} y \) and \( y \mathcal{O} x \) then \( x = y \);
  iii) transitive: if \( x \mathcal{O} y \) and \( y \mathcal{O} z \) then \( x \mathcal{O} z \);

The set on which the partial order relation is defined is called partially ordered set. For example, let \( S \) be the set of natural numbers and let the relation be \( \leq \). It can be easily seen that the relation \( \leq \) is a partial order. The relation \( \subseteq \) over the set \( S \)
of sets $A, B, \cdots$ also defines a partial order.

- Let $S$ be a partially ordered set of elements $a, b, \cdots$. If $a < b$ and if there is no element $x \in S$ such that $a < x < b$ then $b$ is said to cover $a$.

Consider the following example. Let $S$ be the set of proper nonempty subsets of $\{a, b, c\}$, i.e.,

$$S = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}\}.$$

Let set inclusion be the ordering relation. The corresponding partial order is given in Fig. 2.8.

\[ \text{Fig. 2.8} \]

Here $D$ covers $B$ and $A$, $F$ covers $B$ and $C$, and so on. Consider another example. Let $S$ be the set of nontrivial factors of 12 and let the ordering relation be $a/b$. The partial order is given in Fig. 2.9. Here 4 covers 2, 6 covers 2 and 3.

\[ \text{Fig. 2.9} \]

Let $xOy$ be one element in the partial order relation $O$. If $x$ and $y$ are interchanged for all $x$ and $y$ the converse of $O$ is obtained. It is denoted by $O'$.

* If a set $S$ is partially ordered by the relation $O$ then it is partially ordered by $O'$. 
The graph corresponding to the set having relation $O'$ can be obtained by reversing the direction of arrows in the graph of $O$. In the above example, the converse of the relation $a/b$ will be $b/a$.

- A node is a descendant or successor of a set of nodes if there is a path from all of them to that node. A node is an ancestor or ancestor of a set of nodes if there is a path from that node to all other nodes in that set. A descendant is called a greatest lower bound or simply g.l.b. of a set of nodes if there is a path from that node to every other descendant of that set. An ancestor is called a least upper bound or l.u.b. of a set of nodes if there is a path from all the ascendants of that set to that particular node.

For a set of nodes g.l.b. and l.u.b. are unique. L.u.b. of $a$ and $b$ is denoted by $a + b$ and the g.l.b. as $a \cdot b$ or simply $ab$.

Consider the following reduced partial order.

![Diagram](image)

Fig. 2.10


Databases and Lattices

37
of \( \{H, E\} = E, \{H, E, G\} = C, \{G, F, B\} = A \), and note that there is no g.l.b. for \( \{B, C\} \) since its successors are \( \{E, F, H\} \) and no successor is a successor of the other two. Similarly, there is no l.u.b. for \( \{E, F\} \).

- A partial order in which every pair of nodes has l.u.b. and g.l.b. is called a lattice. Reduction of a lattice is called Hasse diagram. L.u.b. and g.l.b. of all nodes in a lattice is called full element (\( \Omega \)) and null element (\( \phi \) or 1) respectively. A node with exactly one outgoing edge is called prime node. Similarly, a node with exactly one incoming edge is called a primitive node.

Normally, a lattice is drawn with the full element at the top and the null element at the bottom. All the arrows point downwards. Without having any ambiguity, a lattice can be drawn without having any arrows shown. The partial orders given below in Fig. 2.11.a and Fig. 2.11.b are lattices. We will refer to them as “pentagon lattice” and “box lattice” respectively. Later, we will be making use of these lattices often to explain different concepts and algorithms.

![Fig. 2.11.a](image1)

![Fig. 2.11.b](image2)
Here $A, B, C, E, F, H, I$ are prime nodes and $A, B, C, G, H, I$ are primitive nodes. But the partial order given below in Fig. 2.10 is not a lattice since nodes $E, F$ do not have a l.u.b. and similarly, nodes $B, C$ do not have a g.l.b..

- As defined earlier a saturated set is a set of attributes which does not determine any attribute outside it. *Generators* are those saturated sets whose intersections generate all the saturated sets except the universal set.

Consider the dependencies $AB \rightarrow C$, $C \rightarrow A$. Saturated sets here are $ABC$, $AC$, $B$, $A$, $\phi$. Generators are $A$, $B$, $AC$. Note that the universal set can always be included when we take the intersection of elements in the generator set.

In the results which follow, $x:y$ means $\text{g.l.b.}(x, y)$ and $x + y$ means $\text{l.u.b.}(x, y)$. The universal element is denoted by 1 and the null element by 0. In a lattice, the following are true[25].

\[
\begin{align*}
&\circ 1 \cdot x = x & 1 + x = 1 \\
&\circ 0 \cdot x = 0 & 0 + x = x \\
&\circ xy = yx & x + y = y + x \\
&\circ x(yz) = (xy)z & x + (y + z) = (x + y) + z \\
&\circ x(x + y) = x & x + xy = x \\
&\circ xx = x & x + x = x
\end{align*}
\]

Thus in set theoretic terms, a lattice is a partially ordered set in which every pair of elements has a unique g.l.b. and l.u.b. within the set. In this thesis, only finite lattices are dealt with. The following are some of the important types of lattices.

- Boolean lattices
- Modular lattices
- Complemented lattices
○ Distributive lattices

- Let $S$ be a set of $n$ elements. Then the set of all subsets of $S$ form a boolean lattice if $\text{l.u.b.}(x, y) = x \cup y$ and $\text{g.l.b.}(x, y) = x \cap y$.

Boolean lattices are of special interest to us and are discussed in later sections. Examples for boolean lattices are given in Fig. 2.12.a and Fig. 2.12.b.

![Fig. 2.12.a](image1)

![Fig. 2.12.b](image2)

- In a distributive lattice, $x \cdot (y + z) = x \cdot y + x \cdot z$. In a modular lattice, whenever $x \geq y$, $x \cdot (y + z) = x \cdot y + x \cdot z$. If for each element $x$ there is a complement $y$ such that $x \cdot y$ is the null element and $x + y$ is the full element, then the lattice is a complemented lattice.

* Every distributive, complemented lattice is a boolean lattice.

**Completion Function and its Dual**

Now, we define a function called completion function and its dual depletion function[45,46]. It is shown that these functions defines a lattice completely.
• A function $C : 2^\Omega \to \text{subset of } 2^\Omega$ having the following properties
  
i) $C(U) \supseteq U$
  
ii) $C(C(U)) = C(U)$
  
iii) $C(\Omega) = \Omega$
  
iv) $U \subseteq V \Rightarrow C(U) \subseteq C(V)$

is called a completion function.

Let $\Omega$ be the set of attributes \{A, B, C\} or simply, $ABC$. Let the dependencies between these attributes be $AB \rightarrow C$ and $C \rightarrow A$. Define the completion of a set as follows. Keep adding all those attributes which are determined by this set until no more addition is possible. Thus

$C(\phi) = \phi$, $C(A) = A$, and $C(B) = B$ since $\phi$, $A$, and $B$ does not determine any attribute outside it.

$C(C) = AC$ since $C$ determines $A$.

$C(AB) = ABC$ since $AB$ determines $C$.

$C(AC) = AC$ since $AC$ does not determine $B$.

$C(BC) = ABC$ since $C \rightarrow A$ and thus $BC \rightarrow A$.

$C(ABC) = ABC$

It can be easily verified that the above function satisfies all the properties of the completion function. Similarly, dual of completion function can be defined as follows.

• A function $D : 2^\Omega \to \text{subset of } 2^\Omega$ having the following properties
  
i) $D(U) \subseteq U$
  
ii) $D(D(U)) = D(U)$
  
iii) $D(\phi) = \phi$
  
iv) $U \supseteq V \Rightarrow D(U) \supseteq D(V)$

is called a depletion function.
Let $\Omega$ be the same set as in the previous example with the same set of dependencies. Define depletion of a set as follows. Keep removing attributes from that set which are determined by the attributes outside it until no more steps are possible.

\begin{align*}
\mathcal{D}(\phi) &= \phi \\
\mathcal{D}(A) &= \phi \quad \text{since } A \text{ is determined by } C. \\
\mathcal{D}(B) &= B \quad \text{since } AC \text{ does not determine } B. \\
\mathcal{D}(C) &= \phi \quad \text{since } AB \rightarrow C. \\
\mathcal{D}(AB) &= B \quad \text{since } C \rightarrow A. \\
\mathcal{D}(AC) &= AC \quad \text{since } B \text{ does not determine either } A \text{ or } C. \\
\mathcal{D}(BC) &= BC \quad \text{since } A \text{ does not determine either } B \text{ or } C. \\
\mathcal{D}(ABC) &= ABC
\end{align*}

It can be easily verified that the above function satisfies all the conditions. Notice that each element in the range of depletion function can be obtained by taking complements of each element in the range of completion function with respect to $\Omega$. For the above example,

\begin{align*}
\text{Range}(C) &= \{\phi, A, B, AC, ABC\} \\
\text{Range}(\mathcal{D}) &= \{ABC, BC, AC, B, \phi\}.
\end{align*}

* Range of a completion function is precisely the set of saturated sets and it defines a lattice.

The physical meaning of the function $C(A) = B$, representing the completion function, is that $B$ is the maximal attribute set functionally determined by $A$. The largest superset determined by $A$ is obviously a saturated set. From the axioms of the completion function, it is clear that the members of the range of the completion function are the saturated sets. In the above example with dependencies $AB \rightarrow C$ and $C \rightarrow A$, the completed sets are $\phi, A, B, AC$, and $ABC$ which are precisely the saturated sets. The corresponding lattice can be obtained by drawing a directed edge from set $A$ to set $B$ if $A \supset B$. Thus we have the following lattice given in
Similarly, range of depletion function also gives out dual of set of saturated sets. Here dual of a set means the complement with respect to the full set. In the above example, dual of $A$ is $BC$, dual of $AC$ is $B$ and so on. Depleted sets for the above example are $ABC$, $BC$, $AC$, $B$, and $\phi$. Dual set of each member gives out saturated sets and thus a lattice corresponding to the given relation. Note that these sets by itself are saturated sets. But they form the dual lattice. Thus

- From a given set of dependencies, the corresponding lattice can be constructed using the completion function.

Consider another set of dependencies $C \rightarrow A$, $D \rightarrow B$, $AD \rightarrow C$, $BC \rightarrow D$. The saturated sets here can be calculated using completion function.

\[
\begin{align*}
C(\phi) &= \phi, & C(A) &= A, & C(B) &= B, & C(C) &= AC, & C(D) &= BD, \\
C(AB) &= AB, & C(AC) &= AC, & C(AD) &= ABCD, & C(EC) &= ABCD, & C(BD) &= BD, \\
C(CD) &= ABCD, \\
C(ABC) &= C(ABD) = C(ACD) = C(BCD) = C(ABCD) = ABCD.
\end{align*}
\]

The saturated sets here are $\phi$, $A$, $B$, $AB$, $AC$, $BD$, $ABCD$ and the corresponding lattice is given below in Fig. 2.14.
Completion function partitions the power set of a set of attributes into equivalence classes with each class having a unique maximal element. The set of these maximal elements is the same as the set of saturated sets.

For the first example above the equivalence classes are \{\emptyset\}, \{A\}, \{B\}, \{C, AC\}, and \{AB, BC, ABC\} and the maximal elements are \emptyset, A, B, AC, and ABC. Similarly, for the second example, the equivalence classes are \{\emptyset\}, \{A\}, \{B\}, \{AB\}, \{C, AC\}, \{D, BD\}, and \{AD, BC, CD, ABC, ABD, BCD, ABCD\} having \emptyset, A, B, AB, AC, BD, and \{ABCD\} as the maximal elements. Similar to the above result we have the following result.

* Depletion function partitions the power set of a set of attributes into equivalence classes having a unique minimal element. The set of these minimal elements forms the dual of the set of saturated sets.

Depleted sets for the first example are \{\emptyset, A, C\}, \{B, AB\}, \{AC\}, \{BC\}, and \{ABC\} and the minimal elements are \emptyset, B, AC, BC, and ABC. They directly
define the pentagon lattice, given in Fig. 2.13. Note that to obtain the original lattice we have to take the dual of the above sets. For the second example we have \{\phi, A, C, D, AB, AD, BC\}, \{AC, ABC\}, \{BD, ABD\}, \{CD\}, \{ACD\}, \{BCD\}, and \{ABCD\} as the partitioned classes and \phi, AC, BD, CD, ACD, BCD, and ABCD as the minimal elements. The lattice is given in Fig. 2.15.

Consider a relation having no dependencies. The range of the completion function will be the power set of \(\Omega\) and we will obtain a boolean lattice. Thus we have the following result.

* Every boolean lattice corresponds to a relation having no dependencies.

Note that every \(n\)-attribute relation having no dependencies will have \(\text{DEG}(R) = \prod_{i=1}^{n} \text{CARD}(D_i)\) where \(D_i\) is the domain of the attribute \(A_i\).

**Completion Graph and its Dual**

Here graph corresponding to the completion function is defined. It is further shown that every completion graph can be used to represent a lattice.
A graph with sets as node labels and having \(2^n\) nodes where \(n\) is the number of distinct attributes occurring in these labels is called a completion graph if it has the following properties.

- Out degree of every node is exactly one.
- \((P_1 \rightarrow P_2) \Rightarrow (P_1 \subseteq P_2)\)
- \((P_1 \rightarrow P_2) \Rightarrow (P_2 \rightarrow P_2)\)
- \(((P_1 \rightarrow P_3 \text{ and } P_2 \rightarrow P_4) \text{ and } (P_1 \subseteq P_2)) \Rightarrow (P_3 \subseteq P_4)\)

where \(P_1 \rightarrow P_2\) means that there is an edge from node \(P_1\) to node \(P_2\) in the graph.

The right hand side of the second condition is equivalent to \(P_1 \cap P_2 = P_2\). The physical meaning of each edge \(P_1 \rightarrow P_2\) in the completion graph is that \(C(P_1) = P_2\).

The following is an example of a completion graph.

![Completion Graph Example](Fig. 2.16)

It is obvious that if we collect the nodes with loops and draw edges from sets to its subsets we obtain the lattice. On the other hand, if the lattice is given we can find the dependencies and then obtain the completed sets and the equivalence classes.
which in turn gives the completion graph. Similarly, a depletion graph can also be defined as follows, and it can be shown that every depletion graph represents a lattice.

• A graph with sets as node labels and having $2^n$ nodes where $n$ is the number of distinct attributes occurring in these labels is called a *depletion graph* if it has the following properties.
  
  o Out degree of every node is exactly one.
  o $(P_1 \rightarrow P_2) \Rightarrow (P_1 \supseteq P_2)$
  o $(P_1 \rightarrow P_2) \Rightarrow (P_2 \rightarrow P_2)$
  o $((P_1 \rightarrow P_3 \text{ and } P_2 \rightarrow P_4) \text{ and } (P_1 \subseteq P_2)) \Rightarrow (P_3 \subseteq P_4)$

Here, the interpretation of the symbol $\rightarrow$ is the same as in the completion graph. The following figure illustrates the above concept.

Notice that if we complement each node label in the completion graph, we obtain the depletion graph.
Boolean Matrices and Lattices

We now proceed to define a closed triangular matrix and some other related matrices and to show that each one of them can be used to define a lattice.

- A boolean matrix is a matrix with boolean literals as elements. A boolean matrix with the following properties is called a closed triangular matrix:
  - all the elements in the first row are 1s
  - all diagonal elements are 1s.
  - rows are closed under direct product

For example, consider the following boolean matrix.

\[
A = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

The above matrix \( A \) is a closed triangular matrix since all elements in the first row are 1s, all diagonal elements are 1s, and the direct product (bit by bit boolean product) of any two rows is again a row in \( A \).

- Every closed triangular matrix represents a lattice completely and vice versa.

First we show, by example, how to obtain the corresponding closed triangular matrix from a lattice. Consider the box lattice given in Fig. 2.14. Construct a \( 7 \times 7 \) matrix in which the node labels of the lattice form both row-headings and column-headings. If a column-heading is a subset of a row-heading, the corresponding matrix element is 1, otherwise it is 0. Row-headings and column-headings start with the largest label, continuing with the next largest and so on, following the lexical order within each group. Then the resulting matrix will be a closed triangular matrix. Note that this matrix is the same as the adjacency matrix of the partial
order of the lattice. In this example we have

\[
K = \begin{pmatrix}
ABCD & AB & AC & BD & A & B & 1 \\
ABC & 1 & 1 & 1 & 1 & 1 & 1 \\
AB & 0 & 1 & 0 & 0 & 1 & 1 \\
AC & 0 & 0 & 1 & 0 & 1 & 0 \\
BD & 0 & 0 & 0 & 1 & 0 & 1 \\
A & 0 & 0 & 0 & 0 & 1 & 0 \\
B & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

It is easy to see that a closed triangular matrix represents a lattice. The first row of the matrix corresponds to the full element \( \Omega \). That the rows are closed under bit wise product implies that every pair of elements has a unique g.l.b. A partial order in which every pair of elements has a unique g.l.b. and the full element \( \Omega \) exists, is a lattice. Now, the Hasse diagram of the lattice can be obtained as follows. Draw nodes corresponding to each row of the matrix. If one row belongs to another, then draw an edge from the node corresponding to the latter row to the node corresponding to the former row. Consider the following example

\[
A = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

The graph corresponding to the above matrix is given below in Fig. 2.18.
By taking the transitive reduction we obtain the Hasse diagram.

Let \( n \) be the number of nodes, \( p \) the number of prime nodes and \( q \) the number of primitive nodes in a lattice. We define the following matrices.

- **A-matrix**: an \( n \times q \) matrix in which the node labels of the lattice form row-headings and primitive node labels form column-headings. If a row-heading is a subset of column-heading the corresponding matrix element is 1, otherwise it is 0.

- **B-matrix**: a \( q \times p \) matrix in which the primitive node labels of the lattice form row-headings and prime node labels form column-headings. If a column-heading is a subset of row-heading the corresponding matrix element is 0, otherwise it is 1.

- **C-matrix**: a \( p \times n \) matrix in which the node labels of the lattice form column-headings and prime node labels form row-headings. If a row-heading is a subset of column-heading the corresponding matrix element is 1, otherwise it is 0.

- **D-matrix**: an \( n \times n \) matrix in which the node labels of the lattice form row-headings and column-headings. If a column-heading is a subset of row-heading the corresponding matrix element is 0, otherwise it is 1.

For our box lattice...
Using these four boolean matrices the following results can be derived.
The matrices $A$, $B$, and $C$ are related by the equations

\[ CA = B_T \]
\[ AB = C_T \]
\[ BC = A_T \]

where bar represents complementation and $T$ represents transposition.

We present an informal proof of the above equations.

The $(i, j)^{th}$ element of matrix $CA$ is 1 if node $i$ is a subset of node $k$ and node $k$ is a subset of node $j$ for some $k$. That is, node $i$ is a subset of node $j$. And by definition of matrix $B$, the $(i, j)^{th}$ element of $B_T$ is 1 if node $i$ is subset of node $j$.

The $(i, j)^{th}$ element of matrix $AB$ is 1 if node $i$ is a subset of node $k$ and node $k$ is not a superset of node $j$ for any $k$. That is, node $i$ is not a superset of node $j$. And by definition of matrix $C$, the $(i, j)^{th}$ element of $C_T$ is 1 if node $j$ is not a subset of node $i$.

We can prove $BC = A_T$ in a similar fashion. The result can be easily verified for our example.

The matrices $A$, $B$, $C$, and $D$ are related by the equations $ABC = A_T A_T = C_T C = D$.

The proof becomes clear if we notice that $AB$ generates the columns of $D$ corresponding to prime nodes. When $AB$ is multiplied by $C$, the entire $D$ is generated. The above result can be easily verified for our example.

Each of the matrices $A$, $B$, $C$, and $D$ defines the lattice completely.

The proof follows from the methods of construction of the lattice given below. The lattice can be constructed from $A$-matrix as follows. If the bitwise addition of $i^{th}$ and $j^{th}$ rows results in the $j^{th}$ row then draw an edge from node $i$ to node $j$. The lattice can be constructed from $B$-matrix as follows. Take the bitwise addition of
the columns of $B$ to produce as many columns as possible. Take the complement of the resulting matrix and transpose it to get the $A$-matrix. The lattice can be constructed from $C$-matrix as follows. If the bitwise addition of $i^{th}$ and $j^{th}$ columns results in the $i^{th}$ column then draw an edge from node $i$ to node $j$. Taking the complement of the $D$-matrix gives the adjacency matrix $K$ of the lattice.

Now we move on to define other representations of lattices.

**Generator Set and Armstrong Relation**

Instead of the set of saturated sets, if we are given the generator set the corresponding lattice can be obtained. Saturated sets can be obtained by taking the intersection of elements in the generator sets and adding the universal element. Thus

- A generator set represents a lattice completely.

For example, consider the generator set \{AC, A, B\}. By taking the intersection we get the pentagon lattice. Now we consider another representation of generator sets.

- Any finite subset of natural numbers represents a unique lattice and vice versa.

Let us illustrate this with the help of an example. Let $G = \{1, 3, 4\}$ be the finite subset of natural numbers. Write each element in the binary form. Then we have $G = \{001, 011, 100\}$. Then assign an attribute to each position and write a bar if there is a 0. Thus $G = \{\overline{A}BC, \overline{A}BC, \overline{A}BC\}$. We can then obtain the generator set \{AB, A, BC\} by collecting the complemented part of each element. Once we have the generator set we obtain the set of saturated sets \{Φ, A, B, AB, BC, ABC\}. Note that we have to add the full set $ABC$ to obtain the saturated sets. Thus we have the following lattice.
It is obvious from the above example that for every finite subset of natural numbers we will have a unique lattice. From the lattice the generator sets and in turn the subset of natural numbers also can be obtained.

A lattice can also be represented by a class of sets. This becomes trivial from the above discussion. Thus

* Any class of closed sets represents a lattice.

The property closed under intersection assures us of g.l.b.. Moreover, if we add the full set into it we obtain the set of saturated sets. The following is an example of such a class.

\[\{a, c\}, \{a\}, \{b\}, \emptyset\]

The above class will give us the pentagon lattice and it corresponds to dependencies \(C \rightarrow A, AB \rightarrow C\).
Now we will construct a table called Armstrong relation which can represent the corresponding lattice uniquely. From the generator set we write the binary form of each element to obtain a unique number. The following example illustrates this in detail. Let the generator set be \{A, B, AC\}. For A we will write 011 (0 for position A, 1 for B and C), for B we have 101, and for AC we have 010. Thus we obtain the numbers 3, 5, and 2 respectively. Then construct the table as follows.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

Note that the first row corresponds to the full element. This table has exactly the same dependencies we started out with. Thus

* An Armstrong relation gives saturated sets and dependencies, and thus the lattice directly.

**Commutative Idempotent Monoid**

This section gives another representation for lattices. We need some definitions first.

- A set \(S\) with an operation \(\cdot\), denoted by \((S, \cdot)\), is called a *commutative idempotent monoid* if for any three elements \(x, y, \) and \(z\) the following is true:
  
  i) \(x \cdot (y \cdot z) = (x \cdot y) \cdot z\)
  
  ii) \(x \cdot e = e\) where \(e\) is the unit element
  
  iii) \(x \cdot x = x\)
  
  iv) \(x \cdot y = y \cdot x\)
It can be easily verified that the set \( \{ABC, BC, B, A, \phi\} \) with the operation \( \bigcap \) is a commutative idempotent monoid.

* Every commutative idempotent monoid uniquely represents a lattice.

The above results follow directly from the fact that the condition idempotent guarantees loops and transitive edges in the corresponding graph and commutativity removes symmetry. Thus we obtain a partial order with the unit element in it which is a lattice. From the pentagon lattice we can construct the above given set directly. To demonstrate the above result consider the following table.

<table>
<thead>
<tr>
<th>e</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>e</td>
<td>e1</td>
<td>e2</td>
<td>e3</td>
</tr>
<tr>
<td>e1</td>
<td>e1</td>
<td>e1</td>
<td>e2</td>
<td>e4</td>
</tr>
<tr>
<td>e2</td>
<td>e2</td>
<td>e2</td>
<td>e2</td>
<td>e4</td>
</tr>
<tr>
<td>e3</td>
<td>e3</td>
<td>e4</td>
<td>e4</td>
<td>e3</td>
</tr>
<tr>
<td>e4</td>
<td>e4</td>
<td>e4</td>
<td>e4</td>
<td>e4</td>
</tr>
</tbody>
</table>

It can be seen that the above multiplication table corresponds to a commutative idempotent monoid. From the table, the lattice can be constructed as follows.

- Draw nodes for each element in the set.
- Draw an edge from node \( i \) to node \( j \) if the entry corresponding to \( i^{th} \) row and \( j^{th} \) column is \( j \) otherwise do not draw an edge at all.
- Remove the loops and reduce the graph to obtain the Hasse diagram.

We will now see how entropy diagrams can be used to represent a lattice.

**Entropy Diagram**

Every lattice can be represented by a corresponding entropy diagram. Consider the following entropy diagram.
The dependencies of the relation corresponding to the diagram can be obtained by collecting all boolean terms of the shaded regions. And thus we arrive at

\[ f = \overline{a}bc + \overline{c}a. \]

The saturated sets of the relation can be obtained as follows. Collect all sets corresponding to the unshaded regions and the outside region. Collect complemented part of each term to obtain the set of saturated sets. For the above example, the sets are \(ABC, ABC, \overline{A}BC, \overline{A}B\tilde{C}, \) and \(\overline{A}B\tilde{C}.\) Taking the complemented parts we get \(\phi, A, B, AC,\) and \(ABC\) as the saturated sets. And in turn we obtain the pentagon lattice. Thus

* An entropy diagram can represent a lattice completely and vice versa.

In the following part we see the connection between boolean algebra and dependencies.

### 2.4.2 Boolean Algebra and Dependencies

The close connection between Boolean algebra[40] and relational databases theory is established in this section. Moreover, Boolean algebra is shown to be
useful in deriving some of the database results\cite{24,28,42}. All the algorithms are explained with the help of examples.

- A set with two operations, denoted by \((B, +, \cdot)\) is called a \textit{Boolean algebra} if the elements of \(B\) satisfies the following properties.

\[
\begin{align*}
\circ & \quad x + y = y + x & x \cdot y = y \cdot x \\
\circ & \quad x \cdot (y + z) = x \cdot y + x \cdot z & x + (y \cdot z) = (x + y) \cdot (x + z) \\
\circ & \quad x + 0 = x & x \cdot 1 = x \\
\circ & \quad \exists y \ni x + y = 0 & x \cdot y = 1
\end{align*}
\]

where 0 and 1 are called additive and multiplicative identities respectively.

Now we define some terms which will be used later.

- A \textit{dependency term} is a product of two or more boolean variables in which exactly one variable appears uncomplemented. A \textit{unate term} is a product of boolean variables in which all variables appears complemented. A \textit{horn term} is either a dependency term or a unate term.

For example, \(\overline{abc}\) is a dependency term, \(\overline{ab}\) is a unate term. Let us see how we can obtain the corresponding lattice from the dependency function. Consider the same pentagon lattice discussed earlier. The saturated sets are \(\phi\), \(A\), \(B\), \(AC\), and \(ABC\). Write \(abc\) for \(\phi\), \(\overline{abc}\) for \(A\), \(\overline{abc}\) for \(B\), \(\overline{abc}\) for \(AC\), and \(\overline{abc}\) for \(ABC\). Taking the complements of these terms we obtain the dependency function \(f\) as

\[
f = ab\overline{c} + \overline{abc} + \overline{abc}
\]

\[
= ac + \overline{abc}
\]

which straight away gives the functional dependencies \(C \rightarrow A\) and \(AB \rightarrow C\). Consider the box lattice given in Fig. 2.14. The saturated sets here are \(\phi\), \(A\), \(B\), \(AB\), \(AC\), \(BD\), and \(ABCD\). So we have

\[
\overline{f} = ab\overline{cd} + \overline{abcd} + ab\overline{cd} + \overline{abc}d + ab\overline{cd} + \overline{abc}d
\]
and the dependency function

\[ f = abcd + ab\bar{c}d + ab\bar{c}\bar{d} + a\bar{b}\bar{c}d + \bar{a}\bar{b}\bar{c}\bar{d} \]

\[ + \bar{a}bcd + \bar{a}b\bar{c}d + \bar{a}bcd + \bar{a}\bar{b}\bar{c}d \]

\[ = a\bar{c} + b\bar{d} + \bar{a}cd + \bar{b}cd \]

which gives us the corresponding functional dependencies \( C \rightarrow A, D \rightarrow B, AD \rightarrow C, \) and \( BC \rightarrow D. \)

In the same fashion, if the dependencies are given, we can obtain the corresponding lattice as follows. Take the minterms in the complement of the dependency function. Collect the complemented part of each term to obtain the saturated sets. Thus

\[ * \text{ From the dependency function, a unique lattice can be obtained and vice versa.} \]

Consider a relation having no dependencies. Using completion function we saw that this relation corresponds to a boolean lattice. The same result can be seen to be true if we consider the dependency function. Since the dependency function of this relation will have \( 2^n \) elements where \( n \) is the number of attributes, we will have \( 2^n \) subsets of \( \Omega \) as the saturated sets which will correspond to a boolean lattice.

**Prime Implicants**

The prime implicants of the dependency function give the entire set of keys along with simple dependencies which are implied by the given set of dependencies.  

Consider the following relation

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_0 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_2 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_3 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>( c_3 )</td>
</tr>
</tbody>
</table>
having the dependencies $AB \rightarrow C$ and $C \rightarrow A$. The corresponding dependency function is $\overline{a}bc + \overline{c}a$. The prime implicants $\overline{a}b$, $\overline{b}c$, and $\overline{c}a$ can be obtained by adding $\overline{a}bc$ to the dependency function. Thus we obtain the horn function

$$h = \overline{a}b + \overline{b}c + \overline{c}a.$$  

The three terms in the horn function represents all the dependencies in that relation, i.e., $AB \rightarrow C$, $BC \rightarrow A$, and $C \rightarrow A$. But the dependency $BC \rightarrow A$ is a logical consequence of the smaller dependency $C \rightarrow A$. The fully complemented terms in the horn function $\overline{a}b$ and $\overline{b}c$ give the keys $AB$ and $BC$. Any relation having the dependencies will have $AB$ and $BC$ as their keys. Thus

* A lattice can be uniquely represented by the corresponding horn function.

Consider another example of a relation corresponding to the box lattice having dependencies $C \rightarrow A$, $D \rightarrow B$, $AD \rightarrow C$, and $B \rightarrow CD$. We have the dependency function

$$f = \overline{c}a + \overline{d}b + \overline{a}dc + \overline{b}cd.$$  

Adding the fully complemented term $\overline{a}b\overline{c}d$, we obtain the horn function

$$h = \overline{c}a + \overline{d}b + \overline{a}dc + \overline{b}cd + \overline{a}b\overline{c}d$$

$$= \overline{a}d + \overline{b}c + \overline{c}d + \overline{c}a + \overline{d}b.$$  

The keys of a relation having the above set of dependencies are $AD$, $BC$, and $CD$.

**Legitimate joins**

In the first section, the terms legitimate join and lossless decomposition are defined. We will now discuss an algorithm to check whether a given join is legitimate or not for a set of dependencies. In other words, the following algorithm checks whether a join dependency is a valid one for a relation or not. We will explain this algorithm using the previous relation having the dependencies $C \rightarrow A$ and
\( AB \rightarrow C \). Let the attribute names be \( X_1, X_2, \) and \( X_3 \) instead of \( A, B, \) and \( C \). Given the dependencies \( X_1X_2 \rightarrow X_3 \) and \( X_3 \rightarrow X_1 \), the problem is to check whether the join \( X_1X_3 \ast X_2X_3 \) is legitimate. The algorithm has the following steps.

- Write the dependency function as

\[
f = \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_3 x_1 + \bar{x}_1 \bar{x}_2 \bar{x}_3
\]

- Take the complement of the dependency function \( f \) and call it \( g \). Thus

\[
g = (x_1 + x_2 + \bar{x}_3)(\bar{x}_1 + x_3)(x_1 + x_2 + x_3)
= x_1 x_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3
\]

- Multiply the function \( g \) by the attributes in the projections of the join and name them \( g_{13}, g_{23}, \) and \( g_3 \) (where \( g_3 \) is the common part of the projections.)

For the projections \( X_1X_3 \) and \( X_2X_3 \),

\[
g_{13} = g(x_1 + x_3)
= x_2 x_3 + x_1 x_3
= x_1 x_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3
\]

\[
g_{23} = g(x_2 + x_3)
= x_2 x_3 + \bar{x}_1 x_2 + x_1 x_3 + \bar{x}_1 x_2 \bar{x}_3
= x_1 x_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3
\]

\[
g_3 = g(x_3)
= x_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 x_3
= x_1 x_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3
\]

- Take the Hamming weight of \( g \) and the projections (Hamming weight of the function is the number of minterms of the equation). Let the Hamming weight of \( g \) be \( H \), that of \( g_{13} \) be \( H_{13} \), that of \( g_{23} \) be \( H_{23} \), and that of \( g_3 \) be \( H_3 \). For this example we have \( H = 4, H_{13} = 3, H_{23} = 4, \) and \( H_3 = 3 \). We
say that if the Hamming weight equation \( H = H_{13} + H_{23} - H_3 \) is satisfied, the given join is legitimate. Note that this Hamming weight equation is a direct observation obtained from the inclusion exclusion principle. Thus the join \( X_1X_3 \ast X_2X_3 \) is legitimate.

The following are some of the corollaries:

\[ * \quad H \geq H_{13} + H_{23} - H_3 \]

\[ * \quad \text{If we consider any other set of dependencies which is a superset of the above set, the same join will again be legitimate.} \]

Now let us check if the join \( X_1X_2 \ast X_2X_3 \) is legitimate.

\[
\begin{align*}
g_{12} &= x_1x_2x_3 + \bar{x}_1x_2x_3 + \bar{x}_1\bar{x}_2x_3 + x_1\bar{x}_2x_3 \\
g_{23} &= x_1\bar{x}_2x_3 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 \\
g_3 &= x_1x_2x_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3
\end{align*}
\]

The Hamming weights \( H = 4, \quad H_{12} = 4, \quad H_{23} = 4, \) and \( H_3 = 3. \) In this case the Hamming weight equation \( H = H_{12} + H_{23} - H_2 \) does not satisfy. Therefore, \( X_1X_2 \ast X_2X_3 \) is not a legitimate join.

**Boolean Algebra and MVDs**

In this section, representation of MVDs in terms of boolean algebra is discussed. The MVD \( ABC \rightarrow DE \) in the relation \( \mathbb{R}(A, B, \ldots, X) \) can be represented by the entropy equation

\[
H(\bar{A}\bar{B}\bar{C}(D + E)(F + \cdots + X)) = 0
\]

(The last factor consists of all those variables which have not appeared in the statement of the MVD). The dependency can be stated as, the variables \( \{DE|ABC\} \) and \( \{F\cdots X|ABC\} \) are independent. In other words, variables \( \{D, E\} \) and \( \{F, \cdots, X\} \) are independent when \( \{A, B, C\} \) is given. Further the entropy of every minterm of

\[ \bar{A}\bar{B}\bar{C}(D + E)(F + \cdots + X) \]
is zero. This follows from a slight generalization of the fact that
\[ H((A_1 + A_2 + \cdots + A_r)(A_{r+1} + \cdots + A_n)) = 0 \]

This implies that \( \{A_1, A_2, \ldots, A_r\} \) and \( \{A_{r+1}, \ldots, A_n\} \) are independent and hence the entropy of every minterm of
\[ (A_1 + A_2 + \cdots + A_r)(A_{r+1} + \cdots + A_n) \]

is zero. From the above statement it follows that the boolean function \( \bar{A}\bar{B}\bar{C}(D + E)(F + \cdots + X) \) can represent an MVD. To fix these ideas we give a simple example of a relation where MVD \( B \rightarrow C \) holds and we show that \( H(\bar{B}AC) = 0 \). The relation is shown below.

<table>
<thead>
<tr>
<th>Rent</th>
<th>A</th>
<th>Ocupant</th>
<th>B</th>
<th>Pet</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td></td>
<td>Nair</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>Mehta</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>Mehta</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td></td>
<td>Taneja</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>Mathur</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>Menon</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td></td>
<td>Chhibber</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assigning the probability of 0.125 for each one of the tuples occurring in the table, we get
\[ H(A + B + C) = 3 \]
\[ H(A + B) = 5/2 \]
\[ H(B + C) = 3 \]
\[ H(B) = 5/2 \]

so that
\[ H(\bar{B}AC) = H(A + B) + H(B + C) - H(A + B + C) - H(B) \]
\[ = 5/2 + 3 - 3 - 5/2 = 0 \]
From the above results we conclude that any set of dependencies can be represented by a boolean function. For example, if $R(A, B, C)$ has the dependencies $A \rightarrow B$, and $C \rightarrow A$ in it, we can write the boolean function as

$$f = \overline{a}bc + \overline{c}a$$

### Boolean Dependencies

We have seen that boolean algebra plays an important role in defining dependencies. But we notice that many boolean functions like $ABCDE$ have not yet been defined. Here we attempt to describe the dependencies corresponding to these boolean terms. We call this kind of dependency a **boolean dependency**. The corresponding dependency for boolean term $\overline{A}BCDE$ is given by $AB \rightarrow CDE$. Now, since we know that the entropy of dependencies is equal to zero, entropy of the above boolean term is $H(\overline{A}BCDE) = 0$. Expanding the left hand side, we get

$$H(A + B + C) + H(A + B + D) + H(A + B + E) - H(A + B + C + D) - H(A + B + C + E) - H(A + B + D + E) - H(A + B) = 0.$$  

Or,


$$= H(A + B + C + D + E) + H(A + B + C) + H(A + B + D) + H(A + B + E)$$

On looking at the relation again, we see that the above system gives natural join equation

$$ABD \ast ABCE \ast ABDE = ABCDE$$

Thus we find that the given boolean dependency can be represented as the join of particular relations. Now, corresponding to zero boolean function i.e., minterm with all terms uncomplemented, as in $ABC$ for instance, we find there are many join dependencies which imply all relations whose join does not give the original relation, i.e., there is a class of joins for which the boolean dependency is zero. Adding the
relation, i.e., there is a class of joins for which the boolean dependency is zero. Adding the zero boolean dependency does not, therefore, alter the original dependency, and corresponding to a boolean function we thus find a class of joins. So, corresponding to every minterm there is an equivalence class of joins i.e., every one of the joins gives this minterm.

In the next section, normal forms of relations are discussed. We have concentrated on the most important form, Boyce-Codd Normal Form (BCNF) and algorithms are given to decompose a relation into BCNF.

2.5 Normal Forms

Data normalization is one of the most important concepts in the design aspect of databases. It is important to keep the relations in normalized forms for better storage and retrieval of data. There are many algorithms available in the literature[5,6,17,21,22,26]. In this section we discuss these normal forms. Moreover, algorithms are given to decompose a relation into BCNF which are based on the lattices and boolean algebra concepts discussed in earlier sections.

- A relation is said to be in first normal form (1NF) if the domain of every attribute in the relation is a simple set. If the determinant of every nonprime attribute is an superkey then, the relation is in second normal form (2NF). If the determinant of every nonprime attribute is a key, then the relation is said to be in third normal form (3NF). A relation is in Boyce-Codd normal form (BCNF) if every determinant is a key.

Note that in relational database theory we consider relations in 1NF only. The other two important normal forms related to MVDs are defined below.

- A relation is in fourth normal form (4NF) if whenever $U \rightarrow V$ is a
is a logical consequence of the set of key dependencies of the schema. A relation is in *projection-join normal form* (PJNF) sometimes called *fifth normal form* (5NF) if every JD is a logical consequence of the set of key dependencies of the relation.

Now we discuss BCNF and give two algorithms to decompose a relation into BCNF.

### 2.5.1 Boyce–Codd Normal Form

In this section we discuss this particular normal form in detail and give some algorithms to decompose a relation into BCNF[16]. A procedure is given to decompose a relation having only functional dependencies into BCNF. This procedure is based on boolean algebra theory discussed earlier.

**Algorithm 1:**

First, we take an example to illustrate the decomposition procedure. Consider a relation in a database in which there are four attributes \{A, B, C, D\} and following dependencies: \(C \rightarrow A\), \(D \rightarrow B\), \(AD \rightarrow C\), \(BC \rightarrow D\). Write down the dependency function \(f\) and reduce it to the minimal boolean function. In this case

\[
f = \bar{c}a + db + \bar{a}dc + \bar{b}cd.
\]

To facilitate the decomposition it is necessary to list all the keys of the relation. As we have seen earlier these are the fully complemented terms in the function

\[
f + \bar{x}_1 \bar{x}_2 \cdots \bar{x}_n
\]

where \(X_i\)'s are the attributes of the relation. Note that if all the terms in the prime implicants are keys then the relation is already in BCNF. The keys of the above relation are \(AD\), \(BC\), and \(CD\).
The BCNF decomposition can be obtained as follows. Choose one of the smallest terms in \( f \). If there is more than one term with the same number of attributes, then choose the term which has the lexically smallest dependent attribute. Here we choose \( CA \). Delete all the terms in \( f \) having all the attributes of this term. Thus

\[
f = \overline{db} + \overline{bcd}.
\]

Delete all the terms from the list of keys having the dependent attribute of the chosen term in it. Here we delete \( AD \). Now choose the next smallest term \( BD \) and delete \( \overline{bcd} \) from \( f \) and \( BC \) from the keys. Since there are no terms left in \( f \) list out the terms which are left in keys and all the chosen terms to obtain the decomposition. Thus a BCNF of the above relation will be \( CD, AC, BD \). It is clear from the above procedure that we cannot choose a term in \( f \) which has a smaller dependency within it. Thus all the chosen terms having, say, \( k \) attributes, will have determinants of size \( k - 1 \) and hence are in BCNF.

From the above example, the general procedure for obtaining a BCNF decomposition of a relation can be given as follows:

- From minimal dependency function list all terms.
- From the prime implicants list all keys.
- Choose one of the smallest terms in the given dependency function making use of lexical order in case of conflicts.
- Delete all the terms in the dependency function having all the attributes of the chosen term.
- Delete all the terms in the list of keys having the dependent attribute of the chosen term.
- Repeat the above steps until no term is left in the dependency function.
- List out all terms which are left in the keys along with the chosen terms.
- Remove all terms which are subsets of other terms listed to obtain a BCNF.
Consider another example of a relation which has the following set of dependencies: \( C \rightarrow A \), and \( AB \rightarrow C \). Note that this corresponds to the pentagon lattice. We have \( f = \bar{c}a + \bar{a}bc \). Adding the fully complemented term \( \bar{a}\bar{b}\bar{c} \), we obtain the keys \( AB \) and \( BC \). By the third step of the above algorithm, choose the smallest term \( AC \). When we choose this term from \( f \) and delete \( AB \) from list of keys, we will exhaust all terms of \( f \) and we obtain the BCNF decomposition \( AC \) and \( BC \).

**Algorithm 2:**

Now we give a procedure for BCNF decomposition based on lattice theory. A relation is said to be in reduced form if all the equivalent attributes are removed. For example, in a relation with dependencies \( A \rightarrow B \) and \( B \rightarrow A \), we retain only one attribute between \( A \) and \( B \). For this algorithm we consider only relations in reduced form. As in the first algorithm we take an example to illustrate the decomposition procedure. Consider a relation in a database in which there are four attributes \( \{A, B, C, D\} \) and the following dependencies: \( C \rightarrow A \), \( D \rightarrow B \), \( BC \rightarrow D \), \( AD \rightarrow C \). The lattice which represents the above dependencies is shown in Fig. 2.21.a. The nodes of the Hasse diagram are designated by the corresponding saturated sets and all the edges in the diagram are assumed to have arrows pointing downwards. We saw in earlier sections that the saturated sets are closed under intersection and they by themselves completely define the lattice.

To facilitate the decomposition it is necessary to relabel the Hasse diagram as follows. Starting from the topmost node, from each node label remove all the attributes which occur in the labels of its children. The resultant diagram is shown in Fig. 2.21.b.
After relabeling, every node with outdegree one will have exactly one distinct attribute. Other nodes may or may not have a label. Now remove all unlabeled nodes except the topmost node by maintaining all existing paths. Here, the middle node can be removed by drawing edges from the topmost node to node $A$ and node $B$. Since there are no paths through the lower most node, it can be removed without drawing any edges. For obtaining the decomposition, all the transitive edges are to be removed. The resultant partial order is shown in Fig. 2.22.
Using Fig. 2.22 it is easy to decompose the relation into BCNF. For each node, collect the labels of its children and add them to its label and list them to obtain the BCNF. Thus a decomposition for our example is \( CD, AC, BD \). Note that we need not include single attributes in the BCNF. The projection corresponding to the topmost node \( CD \) is a key of the relation, hence obviously in BCNF. In every other projection with two attributes there will be one determinant which is the key of that particular relation. In all other projections with more than two attributes, there will be two determinants: one is the parent node label and the other is the set of all its children which determines the parent node. Hence all the projections obtained will be in BCNF.

From the above example, the general procedure for the decomposition of a relation can be given as follows:

- From the saturated sets of the relation, draw the Hasse diagram of the lattice.
- Starting from the topmost node, from each node label remove all the attributes which occur in the labels of its children. Redraw the Hasse diagram.
- Remove all unlabeled nodes except the topmost node, by maintaining all existing paths.
- Remove all transitive edges to obtain a partial order.
- List the projections corresponding to each node after adding all the labels of its children to its label to obtain a BCNF decomposition.

Finally, if we observe that in each projection the key is in its parent projection, it becomes clear that the join of our decomposition will always give the original relation back. The decomposition is obviously unique.

It can be easily verified that this algorithm works for the pentagon lattice too. After relabeling and reducing the Hasse diagram we get the following partial order.
For each node, collecting the labels of its children, we obtain the Boyce–Codd Normal Form decomposition as $BC$ and $AC$.

In this chapter we have seen how relational database theory is closely connected with lattices. We have also given two algorithms to find a BCNF decomposition of a relation. Though we have considered only reduced relations in the case of lattice algorithm, we can extend this algorithm to accommodate nonreduced relations as well. From the discussion it should be clear that these discrete structures can be extensively used in the study of databases.