Hindawi
Mobile Information Systems
Volume 2021, Article ID 5567234, 23 pages
https://doi.org/10.1155/2021/5567234

Research Article

N-Tier Soft Set Data Model: An Approach to Combine the Logicality of SQL and the Flexibility of NoSQL

Jiang Wu,[1] Du Ni,[1] and Zhi Xiao [1,2]

[1]*School of Economics and Business Administration, Chongqing University, Chongqing 400044, China*
[2]*Chongqing Key Laboratory of Logistics, Chongqing University, Chongqing 400044, China*

Correspondence should be addressed to Zhi Xiao; xiaozhicqu@163.com

Received 9 February 2021; Revised 15 May 2021; Accepted 11 June 2021; Published 28 June 2021

Academic Editor: Nhu-Ngoc Dao

Copyright © 2021 Jiang Wu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

To process a huge amount of data, computing resources need to be organized in clusters that can be scaled out easily. Still, traditional SQL databases built on the relational data model are difficult to be put to use in such clusters, which has motivated the movement named NoSQL. However, NoSQL databases have their limits by using their own data models. In this paper, the original soft set theory is extended, and a new theory system called n-tier soft set is brought up. We systematically constructed its concepts, definitions, and operations, establishing it as a novel soft set algebra. And some features of this algebra display its natural advantages as a data model which could combine the logicality of the SQL model (also known as the relational model) and the flexibility of NoSQL models. This data model provides a unified and normative perspective logic for organizing and manipulating data, combines metadata (semantic) and data to form a self-described structure, and combines index and data to realize fast locating and correlating.

1. Introduction

1.1. Background. After entering the 21st century, with the outbreak of Internet applications, the total amount and complexity of digital information possessed by human beings have witnessed an explosive increase at an unprecedented speed, showing many new features. Some professionals believe that we have entered into the era of big data [1, 2]. Databases, as the core part of information infrastructures, play a key role in this historical change. However, relational databases, which previously dominated the market, begin to appear inadequate to cope with some problems of big data [3].

In order to quickly process large volume, fast flowing, and complex data in a limited time to generate value, more computing resources must be acquired. There are usually two schemes: scale up and scale out.

Scale up means configuring better performance hardware for a single computer, such as more and stronger CPUs, and larger and faster memories and disks, but without increasing the number of computers. However, the performance of computer hardware that can be obtained from the market in a period of time has its up limit, and the performance-price ratio of high-end products is usually low, which incurs high cost.

By increasing the number of computers rather than the performance of single computer, scale out incorporates a large number of high cost-effective, low (or mid)-end computers into a cluster to increase computing power. That not only reduces costs in comparison but also makes the cluster more resilient, namely, even if some of the computing nodes failed, the entire cluster can continue to provide services.

1.2. The Challenges of Current Solutions

1.2.1. Relational Databases. However, the relational data model [4] and RDBMS are basically designed for single machine environments. They are not suitable for the case of cluster [3, 5]. From the data model level, relational databases organize the data base on the relational model and use tuples as the record units.

Firstly, according to the definition of the relation model and its normalization theory, a tuple is an ordered list of atomic values that cannot be nested or contain collection types (set, list, and so on) which is difficult to represent complex structures, but there is no such restriction on variables used by application programming, thus resulting in an "impedance mismatch" (a metaphor for the mismatch between the data forms of the relational data model and the application programming model). At present, this problem is usually adjusted by using the middle layer called ORM (object relational mapping).

Secondly, the relational model uses normalization to reduce redundancy and avoid exceptions and ensure the integrity of databases. In a relational database that follows the third (or higher) normal form, the data involved in an unit process of an application are typically scattered across different tables. In order to ensure the ACID (refers to the four basic elements of the correct execution of a database transaction, namely, atomicity, consistency, isolation, and durability) requirements of a transaction and the integrity constraints required by the normal form, a series of locks and resources are costed. In a situation of high concurrency or huge volume, that can egregiously affect the performance and availability of the database.

Moreover, the relational model is algebraically based on relation rather than mapping, which cannot express index by itself (while a mapping is a natural abstraction of an index in mathematics). That renders indexes are external structures, separated from data in implementation, which not only increases the demand of storage space but also makes data difficult to locate each other on their own. To correlate the data between different tables, it is necessary to write complex SQL queries and use expensive Join operation. And in order to support Join operation between tables, the related tables must be placed in a same node, which is not conducive to data dispersion in cluster and usually needs manual design for sharding, making relational databases difficult to scale out.

Meanwhile, a relational database needs a rigid predefined schema. One has to predefine the structures and constraints of tables. And the schema is very difficult to change in reality, falling short of dealing with changing sources and requirements.

1.2.2. NoSQL Databases. Those problems of relational databases have motivated the development of some database products called NoSQL and inspired a new round of innovation for database theory and practice [3, 6, 7]. Different NoSQL products try to solve problems of relational databases from different aspects. According to the data models they use, NoSQL products can be divided into four main types: key-value store, column family, document, and graph [7, 8]. Except for graph databases using graph as a data model, the data models of the first three are based on key-value structures. Key-value store databases are composed of simple key-value pairs, column family databases organize data into two-levels (or more) key-value mappings by row keys and column keys, etc., and document databases organize key-values into documents with accessible internal structures that can be nested with each other.

The main reason why these databases convert the view of data from relations to key-value structures (including simple key-value, column, and document) is for dealing with aggregates. Unlike tuples in relational databases, aggregates are usually designed and used by upper applications (not by databases). It organizes all the data needed in a single processing unit to be accessed together, eliminating expensive and complex SQL queries and table Joins. Aggregates, as natural and independent data distribution units, also make data dispersed easily in a cluster. The form of aggregate is also free, which can easily add or delete content. So, impedance mismatch can be solved without ORM intermediate layers.

Although key-value typed databases have partly solved some problems of relational databases, they do not have rigorous mathematical foundations and there is no connectivity between aggregates, resulting in the difficulties of complex querying and understanding connections among data. On the other hand, relational databases with rigorous and precise algebraic foundation may use a powerful query language based on relational algebra to analyze and reason data freely and logically in the case of a small amount of data on a single machine. However, in the case of big data or in a cluster, it is also difficult to dig out value from the connections among data by using Join operation. So, the graph database, based on graph theory, is designed to explore the connections among data expediently. The graph model represents data as a set of nodes, node attributes, and edges, providing fast and efficient performance of traversing the whole graph with index-free adjacency. However, the graph model focuses on connections and networks, and it is not good at expressing entity and its attribution (mathematically, nodes in a graph have no attributes, and on the implementation, simple key-value pairs are used to store attributes), so it has a specialized range of application and lack of generality [9, 10].

At present, the database models used by the mainstream are the relational model (SQL) and NoSQL (key-value, column family, file, graph, etc.) model. They are proposed to solve the problem that the relational model is too rigid to change the database schema (especially in vast amounts of data) and difficult to distribute. However, the new NoSQL models sacrifice the mathematical rigor of the relational model and the freedom of query expression.

A model that combines the same mathematical logic foundation as the relational model and uses a key-value class data structure urgently requires studying. It can be easy to distribute and also change the mode. We think that this improvement can use the "key-value pair" data structure in a distributed environment to realize a database with rigorous algebraic logic, which combines the advantages of SQL and NoSQL, and has a specific practical significance.

1.3. Our Approach. All these problems motivate us to explore a new data model which will not only maintain the merits of key-value structures, lend data the ability to

describe itself, and can be easily located and moved in a cluster but also have an appropriate normalization and a rigorous algebraic basis like the relational model that can enable a powerful query language independent of products to be applied freely and logically. At the end, we focused on an algebraic theory called soft set. Soft set theory is a mathematics theory proposed by Russian mathematician Molodstov in 1999 in order to solve uncertainty problems. The basic idea is to provide semantic parameterized sets by using a generalized set-value mapping [11].

Just because a soft set is a mapping that allows fuzzy semantics for its parameters and sets for its return values, and mappings in mathematics has natural connection with key-value structures, and sets as return values can have internal structures that can be manipulated, we finally saw the hope that soft set could be used as a mathematical abstraction for an intricate key-value structure [10, 12–15].

Molodstov gave the initial definition of soft set and a general operation and introduced several possible applications in [11]. Maji et al. studied the theory of soft sets in more details [16], introduced the concepts of subset, intersection, union, and complement of soft sets, and discussed their properties (but Yang and Ali et al. pointed out that these properties were incorrect and improved them [17, 18]). Subsequently, a variety of operations and algebraic properties of soft set have been proposed and studied [18–21]. Original soft set has been extended by combining it with other uncertainty theories such as fuzzy set and rough set [22–31], and by using algebraic properties of soft set, new algebraic structures have been constructed [21, 32–36]. Cagman and Enginoğlu gave a new definition of soft set in a form of the extension of set-valued mapping which is different from the original one. Base on that, several related operations have been proposed, a new theory system has been constructed, and a new decision-making method has been presented [37]. At present, soft set theory is widely used in parameter reduction and decision making [38], and a large number of methods for parameter reduction [39–43] and decision making [44–46] have been developed.

In the second section, we will review the soft set theory. Because previous soft set theories are not suitable to be the algebraic basis of the data model we need, we will extend the original soft set theory from the basic structure and systematically introduce a new soft set algebra called n-tier soft set, including its definitions, operations, and related concepts, which will form a complete system and provide the theoretical basis for the later data model. In the third section, we will illustrate why and how to use n-tier soft set to build a data model, define the infrastructure and modeling principles, and finally, explain its features and advantages.

2. N-Tier Soft Set Theory

2.1. The Definitions of N-Tier Soft Set. Before defining n-tier soft set, we first review the basic definition of soft set.

Definition 1. Let a nonempty set Ube a universal set and $\mathrm{P}(U)$be a power set of U. Let a nonempty set Xbe a parameter set. Then, $\mathbf{F}$is called a soft set if and only if $\mathbf{F}$is a mapping from $X$to $\mathrm{P}(U)$:

$$\mathbf{F}\colon X \longrightarrow \mathrm{P}(U). \tag{1}$$

This definition is slightly different from Molodtsov's initial one [11], and it is more similar to Cağman's definition [37]. Generally, we prefer to define a soft set as a special mapping directly rather than an ordered pair consists of a mapping and a parameter set.

A mapping also can be treated as a set of ordered pairs, so an equivalent definition is given.

Definition 2. Let a nonempty set Ube a universal set and $\mathrm{P}(U)$be a power set of U. A nonempty set $X$is called a parameter set and $X \times \mathrm{P}(U) = \{(x, Y) | x \in X \wedge Y \in \mathrm{P}(U)\}$is a Cartesian product of $X$and $\mathrm{P}(U)$, $\mathbf{F}$is called a soft set if and only if $\mathbf{F} \subset X \times \mathrm{P}(U)$, and each $x \in X$appears and appears only once as the first item in an ordered pair, which is

$$\mathbf{F} \subset X \times \mathrm{P}(U) \wedge \forall x \in X\colon \exists! Y \in \mathrm{P}(U)\colon (x, Y) \in \mathbf{F}. \tag{2}$$

Example 1. Examples for soft set: let $A = \{1, 2, 3, 4\}$, $B = \{a, b, c\}$, and a possible soft set on $B$to $A$is $\{(a, \{1, 2, 3\}), (b, \{3, 4\}), (c, \varnothing)\}$. Let $A = \mathbb{R}, B = \mathbb{R}$, and a possible soft set is $\{(x, \{y | y \in \mathbb{R} \wedge y - x > 0\}) | x \in \mathbb{R}\}$.

The definitions above point out that mapping and set are two equally views of soft set. So, for soft set, general properties and operations about set are also suitable (for example, intersection, union, and complement in the sense of a general set). However, the results of these operations may not be enclosed in soft set (like the union operation $\cup$under general sets may destroy mapping condition of soft sets). When applying these general set operations, we treat soft set as a general set directly. In addition, in the following discussion, we will frequently apply both mapping operations and set operations on soft sets to avoid introducing too many notations. For example, there are two soft sets $\mathbf{F}, \mathbf{G}\colon X \longrightarrow \mathrm{P}(U)$. $\mathbf{F}(x)$is the image of x(an element in X) under the mapping rule by soft set $\mathbf{F}$, while $\mathbf{F} \cap \mathbf{G}$is the intersection of two soft sets as the sets of ordered pairs. And $(\mathbf{F} \cap \mathbf{G})(x)$is the image of $x$by the intersection (noticed that the intersection of two soft sets preserves a mapping).

Those notations are concise and enable us to see an important property of soft sets clearly, that is, the ability to maintain mapping after some splitting, merging, or deformation operations.

Meanwhile, because a soft set can be seen as a set-valued mapping, and we also can consider a soft set as a set. Such definition provides a crucial recursive way to construct a new structure, which furnishes the soft set theory with a new and richer content. Next, we will introduce a new notation $\mathbb{S}$to represent a kind of sets of soft sets and define n-tier soft set.

Firstly, we define n-tuple, n-ary Cartesian product, and some other related concepts and introduce some notations to facilitate the following discussion.

Definition 3. An n-tuple is a finite ordered list of $n$elements, where $n$is a positive integer. Formally,

$$(x_n,\ldots,x_1) \coloneqq \begin{cases} x_1, & n=1,\\ \{\{x_2\},\{x_2,x_1\}\}, & n=2,\\ (x_n,(x_{n-1},\ldots,x_1)), & n>2. \end{cases} \tag{3}$$

And let $\mathbf{x}=(x_n,\ldots,x_1)$, $\mathbf{y}=(y_m,\ldots,y_1)$, $(\mathbf{x}_\mathbf{y})=(x_n,\ldots,x_1,y_m,\ldots,y_1)$be the concatenation of $\mathbf{x}$and $\mathbf{y}$, which is noncommutative and associative, namely, $(\mathbf{x}\_(\mathbf{y}\_\mathbf{z}))=((\mathbf{x}\_\mathbf{y})\_\mathbf{z})=(\mathbf{x}\_\mathbf{y}\_\mathbf{z})$.

In this paper, we use $n(\mathbf{x})$to denote the arity of $\mathbf{x}$. Let $i\in\{1,\ldots,n(\mathbf{x})\}$, $\mathbf{x}[i]$denote the i-th component from the right to the left in tuple $\mathbf{x}$. $\mathbf{x}[\backslash i]$is used to denote the new tuple obtained from the tuple $\mathbf{x}$by removing the i-th component from the right to the left.

Definition 4. Let $\mathbf{X}=(X_n,\ldots,X_1)$be an n-tuple which is composed of $n$sets, the n-ary Cartesian product is defined as follows:

$$\Pi\mathbf{X}=\Pi(X_n,\ldots,X_1)\coloneqq\begin{cases} X_1, & n=1,\\ \{(x_2\_x_1)|x_2\in X_2\wedge x_1\in X_1\}, & n=2,\\ \Pi(X_n,\Pi(X_{n-1},\ldots,X_1)), & n>2. \end{cases} \tag{4}$$

Using the usual notation $\times$, it also can be denoted as $X_n\times\cdots\times X_1$. The n-ary Cartesian product defined here is flat, noncommutative, and associative. Namely, that let X,Y,Zbe three sets: $X\times Y\neq Y\times X$, $X\times(Y\times Z)=(X\times Y)\times Z=X\times Y\times Z$. $\mathbf{X}=(X_n,\ldots,X_1)$is called underlying sets of the nCartesian product $\Pi\mathbf{X}$, and the subset of n-ary Cartesian product is called an n-ary relation.

In particular, when $n=1$, then $\Pi(X_1)=X_1$, so the unary Cartesian product with only one set is equal to the set itself. Its elements and subsets are called unary tuple and unary relation, respectively ($(x_1)$and $x_1$are different representations of the same element). And if $\exists X_i=\varnothing, i=1,\ldots,n$, then we can get $\Pi(X_n,\ldots,X_1)=\varnothing$from the definition directly.

Definition 5. N-tier soft set: let $\mathbf{U}=(U_n,\ldots,U_1)$be an n-tuple consisting of nonempty domains.

When $n=1$, we define

$$\mathbb{S}\mathbf{U}=\mathbb{S}(U_1)\coloneqq\mathrm{P}(U_1). \tag{5}$$

Among which, $\mathrm{P}(X)$refers to the power set of X, that is, the set of all subsets of X.

When $n>1$, we define

$$\mathbb{S}\mathbf{U}=\mathbb{S}(U_n,\ldots,U_1)\coloneqq[U_n\longrightarrow\mathbb{S}(U_{n-1},\ldots,U_1)]. \tag{6}$$

Among which, $[X\longrightarrow Y]$refers to the set of all mappings from the domain set $X$to the codomain set Y.

When $n>1$, any element in $\mathbb{S}(U_n,\ldots,U_1)$, which can be a mapping $\mathbf{F}\colon U_n\longrightarrow\mathbb{S}(U_{n-1},\ldots,U_1)$, is called an n-tier soft set about $(U_n,\ldots,U_1)$.

$(U_n,\ldots,U_1)$is called the underlying domains of $\mathbf{F}$, denoted as $\mathrm{und}(\mathbf{F})$.

In this paper, $n(\mathbf{F})$refers to the arity of soft set $\mathbf{F}$, that is, $n(\mathbf{F})\coloneqq n(\mathrm{und}(\mathbf{F}))=n(\mathbf{U})=n$. And $\mathrm{dom}(\mathbf{F})=U_n$, $\mathrm{cod}(\mathbf{F})=\mathbb{S}(U_{n-1},\ldots,U_1)$, and $\mathrm{ran}(\mathbf{F})=\{\mathbf{F}(x)|x\in U_n\}$are the domain, codomain, and range of $\mathbf{F}$, respectively.

When $n=2$, then any element in $\mathbb{S}(U_2,U_1)$, which can be a mapping $\mathbf{F}\colon U_2\longrightarrow\mathbb{S}(U_1)$, is a binary soft set defined in Definition 1.

When $n=1$, then $\mathbf{F}\in\mathrm{P}(U_1)$, so $\mathbf{F}$degenerates into a subset of U_1. Following the name of unary relation, we call it unary soft set, and the underlying domain of it is a unary tuple, that is, $\mathrm{und}(\mathbf{F})=U_1$.

Example 2. $\mathbb{S}(C,B,A)\coloneqq[C\longrightarrow[B\longrightarrow\mathrm{P}(A)]]$is a set consisting of ternary soft set whose underlying domains are (C,B,A).

Next, we will define some other important concepts related to soft set.

Definition 6. Soft empty set $\widetilde{\varnothing}$: let $n$be a positive integer and $\mathbf{U}=(U_n,\ldots,U_1)$be an n-tuple consisting of nonempty domains. $\widetilde{\varnothing}\,\mathbf{U}$is called a soft empty set of $\mathbb{S}\mathbf{U}$if and only if

$$\widetilde{\varnothing}\,\mathbf{U}=\widetilde{\varnothing}\,(U_n,\ldots,U_1)\coloneqq\begin{cases} \varnothing, & n=1,\\ f\colon \begin{matrix} U_n & \longrightarrow\mathbb{S}(U_{n-1},\ldots,U_1),\\ x & \mapsto\widetilde{\varnothing}\,(U_{n-1},\ldots,U_1), \end{matrix} & n>1. \end{cases} \tag{7}$$

Among which,

$$f\colon \begin{matrix} X\longrightarrow Y,\\ x\mapsto f(x), \end{matrix} \tag{8}$$

refers to a mapping f, whose domain and codomain are $X$and Y, respectively, and it maps x, an element of X, to $f(x)$. Sometimes, we simply denote it as follows:

$$f \in [X \longrightarrow Y]\colon x \mapsto f(x). \tag{9}$$

Definition 7. Soft universal set $\tilde{\mathrm{O}}$: let $n$be a positive integer and $\mathbf{U} = (U_n, \ldots, U_1)$be an n-tuple consisting of nonempty domains. $\tilde{\mathrm{O}}\mathbf{U}$is called the soft universal set of $\mathbb{S}\mathbf{U}$if and only if

$$\tilde{\mathrm{O}}\mathbf{U} = \tilde{\mathrm{O}}(U_n, \ldots, U_1) \coloneqq \begin{cases} U_1, & n = 1, \\ f \in \mathbb{S}\mathbf{U}\colon x \mapsto \tilde{\mathrm{O}}(U_{n-1}, \ldots, U_1), & n > 1. \end{cases} \tag{10}$$

Definition 8. Soft subset $\widetilde{\subset}$: let $\mathbf{F}, \mathbf{G} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains and we call $\mathbf{F}$a soft subset of $\mathbf{G}$, denoted as $\mathbf{F} \widetilde{\subset} \mathbf{G}$, if and only if

$$\mathbf{F} \widetilde{\subset} \mathbf{G} \coloneqq \begin{cases} \forall x \in U_1\colon x \in \mathbf{F} \Longrightarrow x \in \mathbf{G}, & n = 1, \\ \forall x \in U_n\colon \mathbf{F}(x) \widetilde{\subset} \mathbf{G}(x), & n > 1. \end{cases} \tag{11}$$

It is important to note here that in earlier soft set theory, the conditions of soft subset can be summarized as follows: $X_1 \subset X_2 \wedge \forall x \in X_1\colon \mathbf{F}(x) = \mathbf{G}(x)$[16]. By regarding soft sets as sets of ordered pairs, this definition means every ordered pair of $\mathbf{F}$is also in $\mathbf{G}$, which can be expressed directly by the subset relation $\subset$ of general sets. However, soft subset is a kind of special inclusion relations of soft set. When the mapping value of soft set is still soft set (rather than a simple set), we compare them by pairs that need recursion as the soft set of values until the mapping values are general sets. In addition, it is also important to note that, at present, we do not consider infinite situation, but only n-tier soft set related to finite n-tuple of domains so all recursive judgments are bound to end. However, how to generalize it to the infinite situation will be discussed in the future study.

Example 3

$$\{(1, \{1, 2\}), (2, \{2, 3\})\} \subset \{(1, \{1, 2\}), (2, \{2, 3\}), (3, \{3, 4\})\} \tag{12}$$

is a subset relation of two soft sets in the sense of general set. The ordered pairs in the first set are all in the second set (but it automatically satisfies the definition of the soft subset at the same time), and

$$\{(1, \{1, 2\}), (2, \{2, 3\})\} \subset \{(1, \{1, 2, 3\}), (2, \{2, 3, 4\}), (3, \{3, 4\})\} \tag{13}$$

is an example of soft subset, because the mapping values determined by the first soft set is subsets of the corresponding values of the second soft set, but none of the elements in the first set is in the second set.

Definition 9. Equality $=$: let $\mathbf{F}, \mathbf{G} \in \mathbb{S}\mathbf{U}$in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains. We consider that $\mathbf{F}$is equivalent to $\mathbf{G}$, denoted as $\mathbf{F} = \mathbf{G}$if and only if

$$\mathbf{F} = \mathbf{G} \coloneqq \begin{cases} \forall x \in U_1\colon x \in \mathbf{F} \Leftrightarrow x \in \mathbf{G}, & n = 1, \\ \forall x \in U_n\colon \mathbf{F}(x) = \mathbf{G}(x), & n > 1. \end{cases} \tag{14}$$

Theorem 1. *Let* $\mathbf{F}, \mathbf{G} \in \mathbb{S}\mathbf{U}$*, in which* $\mathbf{U} = (U_n, \ldots, U_1)$*is an n-tuple consisting of nonempty domains, then* $\mathbf{F} = \mathbf{G}$*if and only if* $\mathbf{F} \widetilde{\subset} \mathbf{G} \wedge \mathbf{G} \widetilde{\subset} \mathbf{F}$.

Proof 1: By using the inductive method, we prove the base case of induction firstly.

According to the definition, when $n = 1$, $\mathbf{F}, \mathbf{G} \in \mathbb{S}(U_1)$, then

$$\begin{aligned} \mathbf{F} = \mathbf{G} &\Longleftrightarrow \forall x \in U_1\colon x \in \mathbf{F} \Longleftrightarrow x \in \mathbf{G} \\ &\Longleftrightarrow \forall x \in U_1\colon (x \in \mathbf{F} \Rightarrow x \in \mathbf{G}) \\ &\wedge (x \in \mathbf{G} \Rightarrow x \in \mathbf{F}) \\ &\Longleftrightarrow (\forall x \in U_1\colon x \in \mathbf{F} \Rightarrow x \in \mathbf{G}) \\ &\wedge (\forall x \in U_1\colon x \in \mathbf{G} \Rightarrow x \in \mathbf{F}) \\ &\Longleftrightarrow \mathbf{F} \widetilde{\subset} \mathbf{G} \wedge \mathbf{G} \widetilde{\subset} \mathbf{F}. \end{aligned} \tag{15}$$

Then, when $n = 1$, the proposition is true.

Next, we prove the inductive step: if when $n = k$, the proposition is true, then, according to the definition, when $n = k + 1$, $\mathbf{F}, \mathbf{G} \in \mathbb{S}(U_{k+1}, \ldots, U_1)$; then,

$$\begin{aligned} &\mathbf{F} = \mathbf{G} \Longleftrightarrow \forall x \in U_{k+1}\colon \mathbf{F}(x) = \mathbf{G}(x), \\ &\forall x \in U_{k+1}\colon \mathbf{F}(x) \in \mathbb{S}(U_k, \ldots, U_1), \\ &\forall x \in U_{k+1}\colon \mathbf{G}(x) \in \mathbb{S}(U_k, \ldots, U_1). \end{aligned} \tag{16}$$

According to the inductive hypothesis,

$$\begin{aligned} &\forall x \in U_{k+1}\colon \mathbf{F}(x) = \mathbf{G}(x) \\ &\quad \Longleftrightarrow \forall x \in U_{k+1}\colon \mathbf{F}(x) \widetilde{\subset} \mathbf{G}(x) \wedge \mathbf{G}(x) \widetilde{\subset} \mathbf{F}(x), \end{aligned} \tag{17}$$

and then

$$\mathbf{F} = \mathbf{G} \Longleftrightarrow \forall x \in U_{k+1}\colon \mathbf{F}(x) \widetilde{\subset} \mathbf{G}(x) \wedge \mathbf{G}(x) \widetilde{\subset} \mathbf{F}(x) \Leftrightarrow (\forall x \in U_{k+1}\colon \mathbf{F}(x) \widetilde{\subset} \mathbf{G}(x)) \wedge (x \in U_{k+1}\colon \mathbf{G}(x) \widetilde{\subset} \mathbf{F}(x)) \Longleftrightarrow \mathbf{F} \widetilde{\subset} \mathbf{G} \wedge \mathbf{G} \widetilde{\subset} \mathbf{F}. \tag{18}$$

So, if the proposition is true when $n = k$, then the proposition is also true when $n = k + 1$. So, according to the induction principle, the proposition is true for any positive integer n, q.e.d. □

Definition 10. Soft power set $\widetilde{\mathrm{P}}$: let $\mathbf{F} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains. Set$\{F' | F' \mathrel{\widetilde{\subset}} \mathbf{F}\}$of all soft subsets of **F**is called the soft power set of **F**, denoted as $\widetilde{\mathrm{P}}(\mathbf{F})$.

It is easy to prove the following properties of soft subset and soft power set by using similar inductive methods in Proof 1.

For any $\mathbf{F} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, then $\mathbf{F} \mathrel{\widetilde{\subset}} \mathbf{F}$, $\widetilde{\varnothing} \mathrel{\widetilde{\subset}} \mathbf{F} \mathrel{\widetilde{\subset}} \widetilde{\mathrm{O}}$, $\mathbf{F} \in \widetilde{\mathscr{P}}(\widetilde{\mathrm{O}})$, $\mathbb{S}\mathbf{U} = \widetilde{\mathscr{P}}(\widetilde{\mathrm{O}})$, and for any $\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3 \in \mathbb{S}\mathbf{U}$, then $\mathbf{F}_1 \mathrel{\widetilde{\subset}} \mathbf{F}_2 \wedge \mathbf{F}_2 \mathrel{\widetilde{\subset}} \mathbf{F}_3 \Rightarrow \mathbf{F}_1 \mathrel{\widetilde{\subset}} \mathbf{F}_3$. The specific proof is similar to Proof 1 and will not be repeated.

2.2. The Operations of N-Tier Soft Set

Definition 11. Soft union $\widetilde{\cup}$: let $\mathbf{F}, \mathbf{G} \in \mathbb{S}\mathbf{U}$in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains and then $\mathbf{F} \mathrel{\widetilde{\cup}} \mathbf{G}$is called the soft union of **F**and **G**if and only if

$$\mathbf{F} \mathrel{\widetilde{\cup}} \mathbf{G} \coloneqq \begin{cases} \mathbf{F} \cup \mathbf{G}, & n = 1, \\ f \in \mathbb{S}\mathbf{U}\colon x \mapsto \mathbf{F}(x) \mathrel{\widetilde{\cup}} \mathbf{G}(x), & n > 1. \end{cases} \tag{19}$$

In addition, let $\mathbf{S} \subset \mathbb{S}\mathbf{U}$, then $\widetilde{\cup}\,\mathbf{S}$is called the soft arbitrary union of **S**if and only if

$$\widetilde{\bigcup}\,\mathbf{S} \coloneqq \begin{cases} \bigcup \mathbf{S}, & n = 1, \\ f \in \mathbb{S}\mathbf{U}\colon x \mapsto \widetilde{\bigcup_{\mathbf{F} \in \mathbf{S}}} \mathbf{F}(x), & n > 1, \end{cases} \tag{20}$$

in which $\widetilde{\cup}_{x \in X} f(x) = \widetilde{\cup}\{f(x) | x \in X\}$.

Definition 12. Soft intersection $\widetilde{\cap}$: let $\mathbf{F}, \mathbf{G} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, then $\mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{G}$is called the soft intersection of **F**and **G**if and only if

$$\mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{G} \coloneqq \begin{cases} \mathbf{F} \cap \mathbf{G}, & n = 1, \\ f \in \mathbb{S}\mathbf{U}\colon x \mapsto \mathbf{F}(x) \mathrel{\widetilde{\cap}} \mathbf{G}(x), & n > 1. \end{cases} \tag{21}$$

In addition, let $\mathbf{S} \subset \mathbb{S}\mathbf{U}$, then $\widetilde{\cap}\,\mathbf{S}$is called the soft arbitrary intersection of **S**if and only if

$$\widetilde{\bigcap}\,\mathbf{S} \coloneqq \begin{cases} \bigcap \mathbf{S}, & n = 1. \\ f \in \mathbb{S}\mathbf{U}\colon x \mapsto \widetilde{\bigcap_{\mathbf{F} \in \mathbf{S}}} \mathbf{F}(x), & n > 1, \end{cases} \tag{22}$$

in which $\widetilde{\cap}_{x \in X} f(x) = \widetilde{\cap}\{f(x) | x \in X\}$.

Definition 13. Soft difference $\widetilde{\setminus}$: let $\mathbf{F}, \mathbf{G} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, then $\mathbf{F} \mathrel{\widetilde{\setminus}} \mathbf{G}$is called the soft difference set of **F**and **G**if and only if

$$\mathbf{F} \mathrel{\widetilde{\setminus}} \mathbf{G} \coloneqq \begin{cases} \mathbf{F} \setminus \mathbf{G}, & n = 1, \\ f \in \mathbb{S}\mathbf{U}\colon x \mapsto \mathbf{F}(x) \mathrel{\widetilde{\setminus}} \mathbf{G}(x), & n > 1. \end{cases} \tag{23}$$

Definition 14. Soft complement $\widetilde{c}$: let $\mathbf{F}, \mathbf{G} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, then $\widetilde{\mathrm{O}}\mathbf{U} \mathrel{\widetilde{\setminus}} \mathbf{F}$is called the soft complement of **F**, denoted as $\mathbf{F}^{\widetilde{c}}$.

Definition 15. Soft symmetry difference $\widetilde{\triangle}$: let $\mathbf{F}, \mathbf{G} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, then $(\mathbf{F} \mathrel{\widetilde{\setminus}} \mathbf{G}) \mathrel{\widetilde{\cup}} (\mathbf{G} \mathrel{\widetilde{\setminus}} \mathbf{F})$is called the soft symmetry difference of **F**and **G**, denoted as $\mathbf{F} \mathrel{\widetilde{\triangle}} \mathbf{G}$.

The above operations of n-tier soft set have the following properties.

Theorem 2. *Let $(U_n, \ldots, U_1)$be an n-tuple consisting of nonempty domains and $\mathbf{F}, \mathbf{G}, \mathbf{H} \in \mathbb{S}(U_n, \ldots, U_1)$. $\widetilde{\mathrm{O}}$and $\widetilde{\varnothing}$ are soft universal set and soft empty set, respectively, in $\mathbb{S}(U_n, \ldots, U_1)$. So, the following properties are true:*

(1) Commutative law:

$$\begin{aligned} \mathbf{F} \mathrel{\widetilde{\cup}} \mathbf{G} &= \mathbf{G} \mathrel{\widetilde{\cup}} \mathbf{F}, \\ \mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{G} &= \mathbf{G} \mathrel{\widetilde{\cap}} \mathbf{F}, \\ \mathbf{F} \mathrel{\widetilde{\triangle}} \mathbf{G} &= \mathbf{G} \mathrel{\widetilde{\triangle}} \mathbf{F}. \end{aligned} \tag{24}$$

(2) Associative law:

$$\begin{aligned} (\mathbf{F} \mathrel{\widetilde{\cup}} \mathbf{G}) \mathrel{\widetilde{\cup}} \mathbf{H} &= \mathbf{F} \mathrel{\widetilde{\cup}} (\mathbf{G} \mathrel{\widetilde{\cup}} \mathbf{H}), \\ (\mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{G}) \mathrel{\widetilde{\cap}} \mathbf{H} &= \mathbf{F} \mathrel{\widetilde{\cap}} (\mathbf{G} \mathrel{\widetilde{\cap}} \mathbf{H}), \\ (\mathbf{F} \mathrel{\widetilde{\triangle}} \mathbf{G}) \mathrel{\widetilde{\triangle}} \mathbf{H} &= \mathbf{F} \mathrel{\widetilde{\triangle}} (\mathbf{G} \mathrel{\widetilde{\triangle}} \mathbf{H}). \end{aligned} \tag{25}$$

(3) Distributive law:

$$\begin{aligned} \mathbf{F} \mathrel{\widetilde{\cup}} (\mathbf{G} \mathrel{\widetilde{\cap}} \mathbf{H}) &= (\mathbf{F} \mathrel{\widetilde{\cup}} \mathbf{G}) \mathrel{\widetilde{\cap}} (\mathbf{F} \mathrel{\widetilde{\cup}} \mathbf{H}), \\ \mathbf{F} \mathrel{\widetilde{\cap}} (\mathbf{G} \mathrel{\widetilde{\cup}} \mathbf{H}) &= (\mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{G}) \mathrel{\widetilde{\cup}} (\mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{H}), \\ \mathbf{F} \mathrel{\widetilde{\cap}} (\mathbf{G} \mathrel{\widetilde{\setminus}} \mathbf{H}) &= (\mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{G}) \mathrel{\widetilde{\setminus}} (\mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{H}), \\ \mathbf{F} \mathrel{\widetilde{\cap}} (\mathbf{G} \mathrel{\widetilde{\triangle}} \mathbf{H}) &= (\mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{G}) \mathrel{\widetilde{\triangle}} (\mathbf{F} \mathrel{\widetilde{\cap}} \mathbf{H}), \\ \mathbf{F} \mathrel{\widetilde{\setminus}} (\mathbf{G} \mathrel{\widetilde{\cup}} \mathbf{H}) &= (\mathbf{F} \mathrel{\widetilde{\setminus}} \mathbf{G}) \mathrel{\widetilde{\cap}} (\mathbf{F} \mathrel{\widetilde{\setminus}} \mathbf{H}), \\ \mathbf{F} \mathrel{\widetilde{\setminus}} (\mathbf{G} \mathrel{\widetilde{\cap}} \mathbf{H}) &= (\mathbf{F} \mathrel{\widetilde{\setminus}} \mathbf{G}) \mathrel{\widetilde{\cup}} (\mathbf{F} \mathrel{\widetilde{\setminus}} \mathbf{H}). \end{aligned} \tag{26}$$

(4) Identity element:

$$\mathbf{F} \mathrel{\widetilde{\cup}} \widetilde{\varnothing} = \mathbf{F}, \mathbf{F} \mathrel{\widetilde{\cap}} \widetilde{\mathrm{O}} = \mathbf{F}, \mathbf{F} \mathrel{\widetilde{\triangle}} \widetilde{\varnothing} = \mathbf{F}. \tag{27}$$

(5) Zero element:

$$\mathbf{F} \mathrel{\widetilde{\cup}} \widetilde{\mathrm{O}} = \widetilde{\mathrm{O}}, \mathbf{F} \mathrel{\widetilde{\cap}} \widetilde{\varnothing} = \widetilde{\varnothing}. \tag{28}$$

(6) Inverse element:

$$\mathbf{F} \mathrel{\widetilde{\triangle}} \mathbf{F} = \widetilde{\varnothing}. \tag{29}$$

(7) Complementary law:

$$\begin{aligned}\mathbf{F}\widetilde{\cup}\mathbf{F}^{\widetilde{c}} &= \widetilde{\mathrm{O}},\\ \mathbf{F}\widetilde{\cap}\mathbf{F}^{\widetilde{c}} &= \widetilde{\varnothing}.\end{aligned} \tag{30}$$

(8) Idempotent law:

$$\begin{aligned}\mathbf{F}\widetilde{\cup}\mathbf{F} &= \mathbf{F},\\ \mathbf{F}\widetilde{\cap}\mathbf{F} &= \mathbf{F}.\end{aligned} \tag{31}$$

(9) Absorption law:

$$\begin{aligned}\mathbf{F}\widetilde{\cap}(\mathbf{F}\widetilde{\cup}\mathbf{G}) &= \mathbf{F},\\ \mathbf{F}\widetilde{\cup}(\mathbf{F}\widetilde{\cap}\mathbf{G}) &= \mathbf{F}.\end{aligned} \tag{32}$$

(10) De Morgan law:

$$\begin{aligned}(\mathbf{F}\widetilde{\cup}\mathbf{G})^{\widetilde{c}} &= \mathbf{F}^{\widetilde{c}}\widetilde{\cap}\mathbf{G}^{\widetilde{c}},\\ (\mathbf{F}\widetilde{\cap}\mathbf{G})^{\widetilde{c}} &= \mathbf{F}^{\widetilde{c}}\widetilde{\cup}\mathbf{G}^{\widetilde{c}}.\end{aligned} \tag{33}$$

By using a similar inductive method in Proof 1, those properties can be proved directly by definition and the specific process will not be repeated here.

Definition 16. Soft range $\widetilde{\mathrm{ran}}$: let $\mathbf{F} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, and then $\widetilde{\mathrm{ran}}(\mathbf{F})$is called the soft range of **F**if and only if

$$\widetilde{\mathrm{ran}}(\mathbf{F}) \coloneqq \begin{cases} \mathbf{F}, & n = 1,\\ \widetilde{\bigcup_{x\in U_n}} \mathbf{F}(x), & n > 1. \end{cases} \tag{34}$$

Definition 17. Key set key: let $\mathbf{F} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, and then key (**F**)is called the key set of **F**if and only if

$$\mathrm{key}(\mathbf{F}) \coloneqq \begin{cases} \mathbf{F}, & n = 1,\\ \{x \in U_n | \mathbf{F}(x) \neq \widetilde{\varnothing}\ (U_{n-1}, \ldots, U_1)\}, & n > 1. \end{cases} \tag{35}$$

Definition 18. Value set val: let $\mathbf{F} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, and then val (**F**)is called the value set of **F**if and only if

$$\mathrm{val}(\mathbf{F}) \coloneqq \begin{cases} \mathbf{F}, & n = 1,\\ \bigcup_{x\in U_n} \mathrm{val}(\mathbf{F}(x)), & n > 1. \end{cases} \tag{36}$$

Definition 19. Selection $\restriction$: let $\mathbf{F} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, and $P(t_n, \ldots, t_1)$is n-ary predicate, so $t_i \in U_i, i = 1, \ldots, n$and $\mathbf{F}\restriction P(t_n, \ldots, t_1)$are called the selection operations if and only if

$$\mathbf{F}\restriction P(t_n, \ldots, t_1) \coloneqq \begin{cases} \{x \in \mathbf{F} | P(x)\}, & n = 1,\\ f \in \mathbb{S}\mathbf{U}\colon x \mapsto \mathbf{F}(x)\restriction P(x, t_{n-1}, \ldots, t_1), & n > 1. \end{cases} \tag{37}$$

Please note that an n-ary predicate is reduced to an $n-1$predicate when its variable is fixed. For example, suppose a 3-ary predicate $P(t_3, t_2, t_1) = (t_3 > t_2) \wedge (t_2 = t_1)$, and when $t_3$takes a fixed value c, $P(c, t_2, t_1) = (c > t_2) \wedge (t_2 = t_1)$becomes a binary predicate with only two variables.

Definition 20. Domain remove $\searrow i$: let $\mathbf{F} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains and $i = 1, \ldots, n$, and $\mathbf{F}[\searrow i]$is called domain remove operation (new soft sets formed by removing the i-th domain of the underlying domain of **F**from right to left and the original mapping relation of **F**), if and only if

$$\mathbf{F}[\searrow i] \coloneqq \begin{cases} \varnothing, & i = 1, n = 1,\\ \mathrm{key}(\mathbf{F}), & i = 1, n = 2,\\ \widetilde{\mathrm{ran}}(\mathbf{F}), & i > 1, n = i,\\ f \in \mathbb{S}\mathbf{U}[\searrow i]\colon x \mapsto \mathbf{F}(x)[\searrow i], & i > 1, n > i. \end{cases} \tag{38}$$

Because of no ambiguity, we use the same token for the n-tier soft set and the n-tuple, and the reader can distinguish them from each other by context.

Definition 21. Domain rise $\langle i$: let $\mathbf{F} \in \mathbb{S}\,\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty

domains and $\mathbf{F}\langle i$is called domain rise operation (new soft sets formed by exchanging the ith and $(i+1)$th domain of $\mathbf{U}$from right to left which is the underlying domain of $\mathbf{F}$and the underlying domain after exchanging is denoted as $\mathbf{U}^*$.), if and only if

$$\mathbf{F}\langle i \coloneqq f \in \mathbb{S}\mathbf{U}^*: \begin{cases} f: y \mapsto \{x | y \in \mathbf{F}(x)\}, & n = 2, i = 1, \\ f: y \mapsto \{(x, \mathbf{F}(x)(y)) | x \in U_n\}, & n > 2, i = n-1, \\ f: x \mapsto \mathbf{F}(x)\langle i, & n > 2, i < n-1. \end{cases} \tag{39}$$

In particular, when $\mathbf{F}$is a binary soft set, the only domain rise $\mathbf{F}\langle 1$is called the reverse of $\mathbf{F}$.

Definition 22. Uncurrying uc: let $\mathbf{F} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains, and $\mathrm{uc}(\mathbf{F})$is called the uncurrying of $\mathbf{F}$if and only if

$$\mathrm{uc}(\mathbf{F}): \begin{array}{l} U_n \times \cdots \times U_2 \longrightarrow \mathrm{P}(U_1), \\ (x_n, x_{n-1}, \ldots, x_2) \mapsto \mathbf{F}(x_n)(x_{n-1}) \ldots (x_2). \end{array} \tag{40}$$

Uncurrying transforms an n-tier soft set into an $n-1$-ary mapping.

Definition 23. Currying cu: let $n$is a definite positive integer, and $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains. $f \in [U_n \times \cdots \times U_2 \longrightarrow \mathscr{P}(U_1)]$and $\mathrm{cu}(f)$are called the currying of $f$if and only if

$$\mathrm{cu}(f) \coloneqq \begin{cases} f, & n = 2, \\ \mathbf{F}: \begin{array}{l} U_n \longrightarrow \mathbb{S}(U_{n-1}, \ldots, U_1) \\ x \mapsto \mathrm{cu}\left(f_{(x)}\right) \end{array}, & n > 2. \end{cases} \tag{41}$$

Mind here, an n-ary function is reduced to an $(n-1)$-ary function when one of its variables is fixed. For example, set $f(x, y) = (x/y)$will be reduced to $f(1, y) = (1/y)$when the value of $x$is fixed. Generally, we obtain the $(n-i)$-ary function which can be denoted as $f_{(c_1,\ldots,c_i)}$by taking $i$values of the n-ary function from left to right, continuously, and then $f_{(c_1,\ldots,c_i)_{(c_{i+1})}} = f_{(c_1,\ldots,c_{i+1})}$.

Currying transforms an $(n-1)$-ary mapping into an n-tier soft set.

Definition 24. Concatenate production $\widetilde{+}$: let $\mathbf{F} \in \mathbb{S}\mathbf{U}$, $\mathbf{G} \in \mathbb{S}\mathbf{V}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains and $\mathbf{V} = (V_m, \ldots, V_1)$is an m-tuple consisting of nonempty domains. If $U_1 = V_m$, then $\mathbf{F} \widetilde{+} \mathbf{G}$is called the concatenate production of $\mathbf{F}$and $\mathbf{G}$if and only if

$$\mathbf{F} \widetilde{+} \mathbf{G} \coloneqq \begin{cases} \mathbf{F} \cap \mathbf{G}, & n = 1, m = 1, \\ \{(x, \mathbf{G}(x)) | x \in \mathbf{F}\} \cup \{(x, \widetilde{\varnothing}\, \mathbf{V}[\backsim m]) | x \notin \mathbf{F}\}, & n = 1, m > 1, \ \mathbf{G}, n > 1, m > 1. \\ \quad f \in \mathbb{S}(\mathbf{U}\_\mathbf{V}[\backsim m]): x \mapsto \mathbf{F}(x) \widetilde{+} \end{cases} \tag{42}$$

Particularly, $\mathbf{F}$is a set when $n = 1$, and $\mathbf{F} \widetilde{+} \mathbf{G}$, directly denoted as $\mathbf{G}(\mathbf{F})$, is also called the restriction of $\mathbf{G}$under $\mathbf{F}$.

Definition 25. Soft direct production $\widetilde{\times}$: let $\mathbf{F} \in \mathbb{S}\mathbf{U}, \mathbf{G} \in \mathbb{S}\mathbf{V}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of nonempty domains and $\mathbf{V} = (V_m, \ldots, V_1)$is an m-tuple consisting of nonempty domains. We call $\mathbf{F} \widetilde{\times} \mathbf{G}$is the soft direct production of $\mathbf{F}$and $\mathbf{G}$if and only if

$$\mathbf{F} \widetilde{\times} \mathbf{G} \coloneqq \begin{cases} \{(x, \mathbf{G}) | x \in \mathbf{F}\} \cup \{(x, \widetilde{\varnothing}\, \mathbf{V}) | x \notin \mathbf{F}\}, & n = 1, \\ f \in \mathbb{S}(\mathbf{U}\_\mathbf{V}): x \mapsto \mathbf{F}(x) \widetilde{\times} \mathbf{G}, & n > 1. \end{cases} \tag{43}$$

Definition 26. Soft mapping production $\widetilde{\circ}$: let $(U_2, U_1), (V_2, V_1)$be a binary tuple consisting of two nonempty domains $\mathbf{F} \in \mathbb{S}(U_2, U_1), \mathbf{G} \in \mathbb{S}(V_2, V_1)$. We call $\mathbf{F} \widetilde{\circ} \mathbf{G}$is the soft mapping production of $\mathbf{F}$and $\mathbf{G}$if and only if

$$\mathbf{F} \widetilde{\circ}\ \mathbf{G} \coloneqq f: \begin{array}{l} U_2 \times V_2 \longrightarrow \mathbb{S}(U_1\_V_1) \\ (x\_y) \mapsto \mathbf{F}(x) \widetilde{\times} \mathbf{G}(y) \end{array}. \tag{44}$$

According to the definition, soft mapping production is associative, namely, $X \widetilde{\circ} (Y \widetilde{\circ} Z) = (X \widetilde{\circ} Y) \widetilde{\circ} Z = X \widetilde{\circ} Y \widetilde{\circ} Z$.

Definition 27. Soft relation: let $\mathbf{F}_1, \ldots, \mathbf{F}_n$be $n$binary soft sets and $\mathbf{R}$is called the soft relation whose underlying domain is $(\mathbf{F}_1, \ldots, \mathbf{F}_n)$if and only if

$$\mathbf{R} \widetilde{\subset} \mathbf{F}_1 \widetilde{\circ} \cdots \widetilde{\circ} \mathbf{F}_n. \tag{45}$$

Definition 28. Associated relation of a soft set: let $\mathbf{F} \in \mathbb{S}\mathbf{U}$, in which $\mathbf{U} = (U_n, \ldots, U_1)$is an n-tuple consisting of non-empty domains. $\mathrm{rel}(\mathbf{F})$is called the associated relation of $\mathbf{F}$if and only if

$$\mathrm{rel}(\mathbf{F}) \coloneqq \begin{cases} \mathbf{F}, & n = 1, \\ \bigcup\limits_{x \in U_n} \{x\} \times \mathrm{rel}(\mathbf{F}(x)), & n > 1. \end{cases} \tag{46}$$

Definition 29. Associated soft set of a relation: let $R \in \mathrm{P}(U_n \times \cdots \times U_1)$be an n-ary relation (if R is an empty set, according to its assumption and context, $R \in \mathrm{P}(U_n \times \cdots \times U_1)$can be considered as an n-ary empty relation whose underlying domain is $(U_n, \ldots, U_1)$. $\mathrm{fun}(R)$is called the associated soft set of a relation of Rif and only if

$$\mathrm{fun}(R) \coloneqq \begin{cases} R, & n = 1, \\ f\colon \begin{array}{l} U_n \longrightarrow \mathbb{S}(U_{n-1}, \ldots, U_1) \\ x \mapsto \mathrm{fun}(\{\mathbf{y} \mid (x, \mathbf{y}) \in R\}) \end{array}, & n > 1. \end{cases} \tag{47}$$

Mind here, we indirectly used inductive definition of tuple. That is, any n-ary tuple could be considered as a nested binary tuple when $n > 1$. For an n-ary relation and a definite value $c \in \mathrm{dom}(R)$, $\{\mathbf{y} \mid (c, \mathbf{y}) \in R\}$is an $(n-1)$-ary relation consisting of $(n-1)$-tuples (if $\{\mathbf{y} \mid (c, \mathbf{y}) \in R\}$is an empty set, it can be seen as an $(n-1)$-ary empty relation).

For mathematics, the n-tier soft set defined in this section and its operations have a wealth of contents to be studied. They have nice properties, soft intersection, soft union, and soft complement, and other operations satisfy all the properties of common set operations (commutation law, association law, etc.). However, this paper does not focus on the discussion of the mathematics. Next, we will focus on explaining why and how to use n-tier soft set as a data model for databases in the era of big data.

3. N-Tier Soft Set Data Model

There is no natural expression for the existence of things or events. Only by purposeful selection, abstraction and simplification can we transform some specific aspects of irregular fields to structured and manipulatable objects. Data model describes the static characteristics and dynamic behavior of database system from the abstract level, providing a logical abstract framework for data representation and operation, and fundamentally determines how data are stored, organized, and manipulated. Therefore, the data model is the core and foundation of the database system, and all database systems must be based on a certain data model. The data model also constitutes a bridge between the upper applications, database system itself, and its underlying physical implementation, which enables them to view and use the data in a unified way.

We have already explained the problems of the relational data model and the most popular NoSQL data models in Introduction. In the second section, n-tier soft set is defined as a nested set-valued mapping that makes it possible to express complex key-value structures. Next, we will set up a new data model by using n-tier soft set algebra.

Just like that we often use a table to represent a relational model, in order to illustrate easily, we will introduce a plain text representation of n-tier soft set at first. It is similar to JSON and independent of specific programming languages, which is called SSSN (soft set serialization notation). The basic construction rules are as follows (just for a demo, the strict definition and parse method will not be discussed in this paper):

(1) Representing strings with double quotation marks, numerical values with literal numbers, and Boolean values with true/false, for example,

"hello World this is SSSN" #String

12345678 #Number

true #Boolean

(2) Representing tuples with contents enclosed in parentheses and separated in comma, for example,

("Joe," "Male," "New York")

(3) Representing sets with contents enclosed in brace and separated in comma, in which the elements cannot be repeated, for example,

{"Elephant," "Monkey," "Zebra," "Panda"}

(4) Representing mappings with contents enclosed in brace, separated in comma and matched by colon (several-to-one ordered pairs, and the left side of colon cannot be duplicated.), for example,

{"name":"Joe," "sex":"male," "address":"New York"}

(5) Representing bijective mappings with contents enclosed in brace, separated in comma and matched by double colon (one-to-one ordered pairs, and neither side of double colon should be duplicated.), for example,

(iii) {"20181001":"Oct−1−2018", "20181031":"Oct−31−2018"}

(6) When colons or double colons are used to pair, the left side of the colon or double colon can only use strings, numbers, Booleans, or tuples, and the right side can use any type of value defined above.

```
{
  ("name," "sex"):{"Joe":"male," "Eva":"Female"},
  ("name," "birthday"):
  {"Joe":"20001001," "Eva":"20000110"}
}
```

In the following discussion, we can see that when we express and pass n-tier soft set in this way, we can not only express and pass the semantics and data integrated by n-tier soft set but also express some important logical constraints.

3.1. Definitions of N-Tier Soft Set Data Model. Next, we will define the n-tier soft set data model.

Definition 30. Domain soft set: let $S$is a finite set of semantic strings and $A$is a finite set of atomic data, then the soft set

$D$is called a domain soft set if and only if it is a soft set from $S$to A:

$$D \in \mathbb{S}(S, A). \tag{48}$$

The elements in $D$are called named domains. The elements in the key set (key(D)) of domain soft set $D$are called domain names. The images of the domain names under the mapping rule of the domain soft set are called the value ranges.

Example 4. Domain soft set:

S = {"name," "sex," "birthday," "telephone," ... }
A = {
"Joe," "male," "20011231," "19990723," "female," "Eva,"
"19980301," "Adam," "Bob," "860702," "13320255520,"
...
}
D = {
"name":{"Joe," "Eva," "Adam," "Bob," ... },
"sex":{"male," "female," ... },
"birthday":{"19870220," "19990723," "19980301,"... },
"telephone":{"860702," "13320255520," "1191101,"... },
...
}

Domain soft set combines semantic and data, determines involved domains, defines the semantic names and value range of a domain, and establishes the finite boundaries of a database system.

Definition 31. Domain relation soft set: let Dbe a domain soft set, $R$is called a domain relation soft set underlying if and only if it is a soft relation of $D$and itself, namely,

$$R \widetilde{\subset} D \widetilde{\times} D. \tag{49}$$

The elements in $R$are called named domain relations. The elements in the key set of domain relation soft set (key(R)) are called domain relation names. The images of the domain relation names under the mapping rule of the domain relation soft set are called the values of domain relations.

Example 5. Domain relation soft set:

R = {
("person," "name"):{"0001":{"Joe"}, "0002": {"Eva"},... },
("person," "sex"):{"0001":{"male"},"0002":{"female"},... },
...
}

By soft mapping production (Definition 26), domain soft set forms a soft relation (Definition 27) with itself: binary tuples are formed between domain names and express the connotation of the relationship, and and binary soft sets are formed between data and express the extension.

Definition 32. Database soft set: let Sbe a finite set of semantic strings and Dbe a domain soft set, then $B$is called a database soft set if and only if $B$is a mapping from $S$to a domain relation soft set underlying D, that is,

$$B\colon S \longrightarrow \widetilde{\mathrm{P}}(D \widetilde{\times} D). \tag{50}$$

The elements in $B$are called named soft set databases. The elements in the key set of B(key(B)) are called database names. The images of the database names under the mapping rule of the database soft set are called the value of such database names.

Example 6. Database soft set:

B = {
"college":{
("person," "name"):{"0001":{"Joe"},... },
("person," "student_id"):{"0001":{"20130201025"},... },
("course," "name"):{"CS001":{"Database"},... },
("course," "credits"):{"CS001":{"3"},... },
...
},
"E−Shopping":{
("person," "name"):{"0001":{"Joe"},... },
("person," "customer_id"):{"0001":{"00302001"},... },
...
},
...
}

Definition 33. Aggregate soft set: let Sbe a finite set of semantic strings, Bbe a database soft set, and $n$be a database name, then $T$is called an aggregate soft set if and only if $T$is a mapping from $S$to $\mathrm{P}(B(n))$, namely,

$$T\colon S \longrightarrow \mathrm{P}(B(n)). \tag{51}$$

The elements in $T$are called soft aggregates.

Example 7. Aggregate soft set: let n = "college," $T\colon S \longrightarrow \mathrm{P}(B(n))$:

T = {
"student":{

```
    ("person," "name"):{"0001":{"Joe"},... },
    ("person," "student_id"):{"0001":
{"20130201025"},... },
    ...
  },
  "course":{
    ("course," "name"):{"CS001":{"Database"},... },
    ("course," "credits"):{"CS001":{"3"},... },
    ...
  },
  ...
}
```

Aggregate soft sets divide a soft set database into different subsets and give each subset a name.

Meanwhile, let Sbe a finite set of semantic strings, Abe a finite atomic data item set, $n \in S$be a database name, and Dbe a domain soft set from $S$to A, and according to the definition, we have

$$\begin{aligned} &D \in \mathbb{S}(S, A), \\ &R \widetilde{\subset} D \tilde{\infty} D \Rightarrow R \in \mathbb{S}(S \times S, A, A), \\ &B\colon S \longrightarrow \tilde{\mathrm{P}}(D \tilde{\infty} D) \Rightarrow B \in \mathbb{S}(S, S \times S, A, A), \\ &T\colon S \longrightarrow \mathrm{P}(B(n)) \Rightarrow T \in \mathbb{S}(S, S \times S, A, A). \end{aligned} \tag{52}$$

Therefore, all the objects defined in this section can be represented as an n-tier soft set consisting of a semantic set $S$and a dataset A. So, the semantics, data, and relations can be fully expressed in an n-tier soft set system without the participation of external information. All the operations defined in Section 2can be directly and conveniently applied to a model or an instance.

For example, for the domain relation soft set Rin Example 5above, we can transform the binary function into a nested soft sets by currying operation (Definition 23) and raise the second domain (Definition 21), which can be denoted as $\mathrm{uc}(R)\langle 2$, as follows:

```
{
  "person":{
    "0001":{"name":{"Joe"},"sex":{"male"},... },
    "0002":{"name":{"Eva"},"sex":{"female"},... },
    ...
  }
}
```

Then, it looks like the BigTable data model proposed by Google in [47], which is a multilevel mapping with domain "person" as row keys, and domain "name," "sex," etc. as column keys. And we can also select some of them to form a new 4-ary soft set by selection in Definition 19or delete a certain key level to make it a 3-ary soft set by domain remove operation in Definition 20. Or to form a deeper, larger n-tier soft set by product operations such as soft direct product (Definition 25), concatenate product (Definition 24), and so on. It should be noted that all these operations are defined recursively just for the rigor of mathematical logic and the convenience of proof and do not imply that they must be implemented by recursive algorithms.

3.2. Modeling with N-Tier Soft Set. Through the above definitions, we get the basic components needed to build the n-tier soft set data model. Next, we use an example to demonstrate the evolution from the relational model to the four popular NoSQL models, then to the n-tier soft set model, to show why and how to use the n-tier soft set data model for modeling.

In the traditional modeling process for relational databases, the initial stage of modeling is to understand conceptual entities in the modeling domain and the relationship between them. Through the discussion between domain experts and system architects and data architects, the results of these understandings often end up forming a so-called conceptual model, which is often represented by an ER diagram. Although the ER diagram is often used in the modeling process of the relational model, it can also provide a common conceptual starting point for all other models in our discussion.

We suppose that we have designed a conceptual model, as shown in Figure 1. It shows a simplified scenario of a common E-shopping site, which contains four entities: customer, order, order item, and product, and represented in a rectangle, respectively. Ellipses connected to an entity with undirected edges represent the attributes of each entity, and the underlined ID attribute uniquely identifies a particular entity. A customer can place multiple orders, each of which contains multiple order items, and each order item relates to a particular product. The relationships between these entities are represented by a diamond with undirected edges, and the quantitative relations are represented by n or 1 on both sides of the diamond. At last, customers can follow each other and know what their friends have bought in this shopping site. We represent the relationship in a diamond with both ends connected to the customer entity, and we use m and n on both sides of the diamond to indicate a multilateral relationship. In the following diagrams, we use rounded rectangles to represent data blocks, which can be atomic (if no internal structure is indicated) or composite (larger rectangles wrapped in other rectangles). Red represents semantics, straight lines represent undirected relations, and arrows represent directed relations.

3.2.1. Modeling with Relational Model. Then, firstly, let us see how to model the scenario with the relational model.

After obtaining a suitable conceptual model, the relational model transforms it into the structures and constraints of tables. As shown in Figure 2, the ID attributes that uniquely identify entities become the primary keys of the tables. The 1 : 1 and n : 1 relations among entities are represented by foreign keys inserted into the corresponding tables (such as customer_id in order table, or order_id in order_item table), and the m : n relationship will be implemented by adding a new relation table.

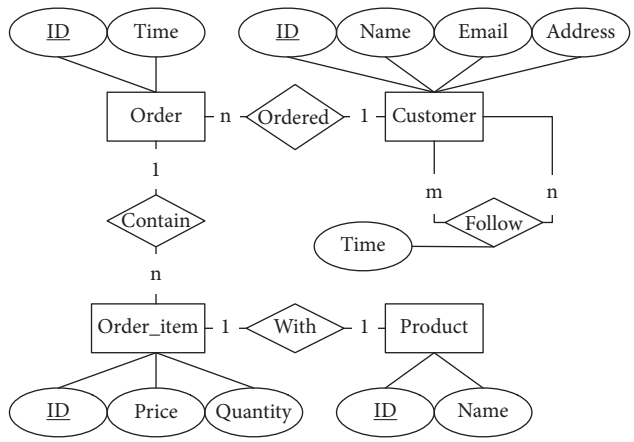

Figure 1: ER model for E-shopping.

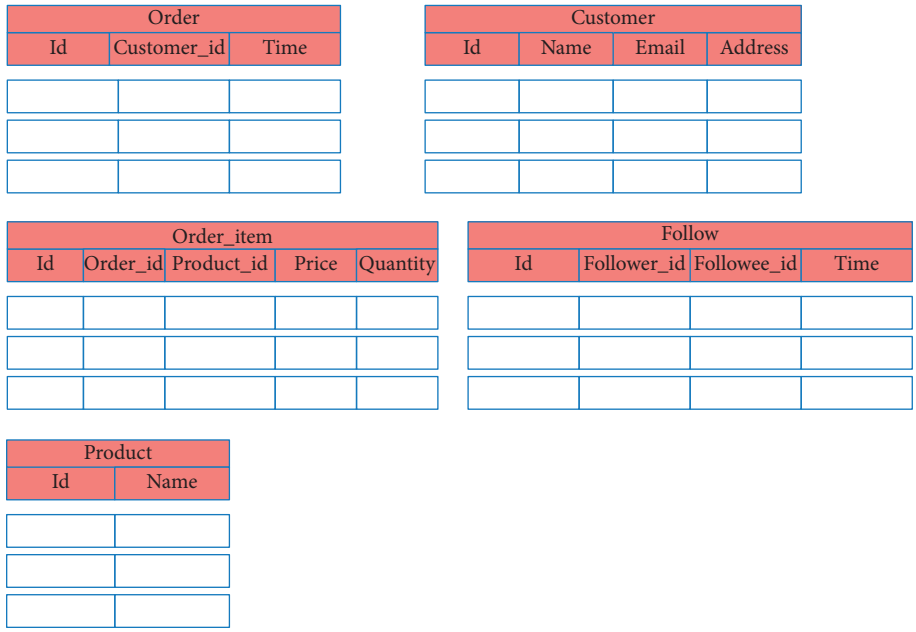

Figure 2: Relational model for E-shopping.

The advantages of relational model lie in its simple and intuitive expression, strict and nice mathematical foundation, and the freedom from the separation of logic and physics. Without any underlying implementation information, a relational database can freely express and obtain information contained in an existing dataset by a small amount of concise operations (relational algebra has been proved to be equivalent to first-order predicate calculus restricted in secure expressions). However, we can also observe several problems with the relational model: as you can see from Figure 2, a table is a regular two-dimensional rectangular array. It consists of tuples that contain the same number of indivisible atomic elements, and a single header provides semantic interpretation for tuples. This form is simple and regular but can lead to the following problems:

(i) Flat: a tuple is a flat and restricted structure, which can only contain indivisible elements. These elements are regarded as atoms at the model level. They have no internal structure and cannot be nested, which restricts its ability to express complex objects and brings the so-called impedance mismatching problem.

(ii) Rigid: in a table, every tuple must contain a same fixed number of elements, and each element is rigidly coupled with its position, so even if there is actually no value in a position, its place shall be filled with the null value.

(iii) Semantic and data separation: table heads as semantics and table bodies as data are separated. In the theory of the relational data model, table names and column names are defined by a metalanguage, and in a specific implementation, a relational database uses a data dictionary separated from the data to store these metadata. That makes it necessary to

process metadata separately before transferring data. This separation of semantic and data makes it difficult to transmit data in a network, while other data formats such as XML or JSON combining semantics with data can enable the transmission of complete information at the same time.

(iv) Index and data separation: the relational model does not express information about how tuples are located or sorted. To find tuples containing certain values in a table, one has to scan and compare them one by one. This renders the relational model too reliable on the external index structure in real use. However, indexing is not a part of the relational model. It not only consumes large storage space but also incurs maintenance costs.

(v) Data and data separation: whether in the same table or between different tables, tuples of relational models are separated from tuples. Their connections which need to be calculated dynamically are implicit in the value of specific data. Conceptually, this shows that the relational model does not directly express the relations between entities. To find links between entities, it is necessary to connect tables with Join operation, which is usually very time-consuming.

3.2.2. Modeling with NoSQL Models. These problems in the relational model have prompted the development of NoSQL data models and database products.

(1) Key-Value Store. Let us first look at the simplest of these: key-value store. The data model of key-value store is very simple. As shown in the Figure 3, the whole database can be divided into two parts: the set of keys on the left and the set of values on the right. We use arrows to indicate the corresponding one-directional access. In our case, we use order id as the keys, and all information related to an order id is placed in its value. The specific content of a value is determined by the upper application, and the database is only responsible for access. Theoretically, key-value store only focuses on the effective access of data, and values are not transparent to the database, which requires users to parse by themselves. If only a part of a value is required, it entails a process of extracting the entire value and filtering out unwanted content, which may be inefficient. So, the column family model and the document model add more internal structures to the values.

(2) Column Family. Logically, a column family model can be regarded as adding a secondary column name to value pairs in the values of a key-value store model, and these secondary pairs can also be grouped into column families. As shown in Figure 4, on the left, the primary keys are also called the row keys, which locate a virtual row. On the right, column name strings (characters enclosed in quotation marks) as secondary keys are located to the values (technically, tertiary keys may also be included, such as time stamps, version stamps, and so on, but skipping them does not affect our discussion). The prefixes in the column name strings divide them into different column families. The column family model can be regarded as a huge sparse two-dimensional table, which is more expressive than the key-value model. And because columns are represented by key-value pairs, they can be added and deleted freely. In our case, like the key-value store model, we also use order id as the row key. However, the value has a richer structure. We store all customer information by customer column family and all order items by order item family and merge product information into them (because product and order item are one-to-one relationships). Different order item information is distinguished by assigning a number to the column key.

(3) Document. The document model has more richer value structure than the column family model. As shown in Figure 5, a document database stores and retrieves all documents as a file cabinet. These documents contain simple key-value pairs (similar to key-value store), nested key-value pairs (similar to the combination of row keys and column keys of column families), lists (returning by sequential numeric subscripts rather than keys), and other nestable contents. This makes the document model even more expressive, and a document can be easily converted into a programming object in an upper application. Like all key-value typed models, the form of documents is flexible, and various structures in documents can be added or deleted freely. In our case, all information of customers, orders, and products is included in a document, which looks like an actual order list.

Generally speaking, all above three models use key-value pairs as basic structures to organize data. Different models use different structural values, which provide different ways of aggregating information.

Key-value pairs are simple but essential. Keys can provide semantics for the values, which uncouple data and their positions, and eliminate the rigidity of system. A key-value pair is a self-described entirety that is no longer dependent on each other in form. At the same time, keys can also help locate values so that they can be accessed quickly. This allows key-value pairs can be easily dispersed into a cluster, and their contents and forms can be very free and flexible. So, we can predetermine all the required content according to the convenience of the upper application and aggregate it together for fast access without Join operation. That partly solves the problems of the relational model. However, key-value typed models also have some problems:

(i) Values can only be accessed one way by keys, and keys cannot be retrieved by values reversely (we can see the directions and granularities of access for different models through the arrows shown in the figures). To find the specific key-value pairs by values, it is necessary to compile external indexes or use external frameworks such as MapReduce for scanning processing.

(ii) There is no connection between key-value pairs. Discrete key-value pairs have many advantages, and they can be formed and operated independently, but

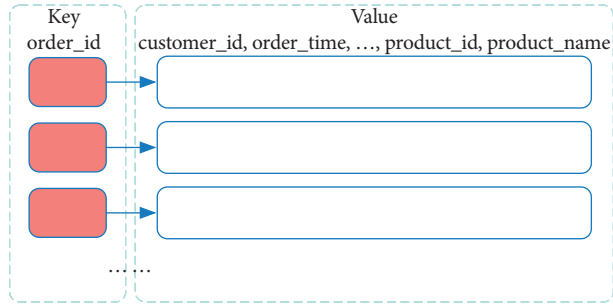

Figure 3: Key-value model for E-shopping.

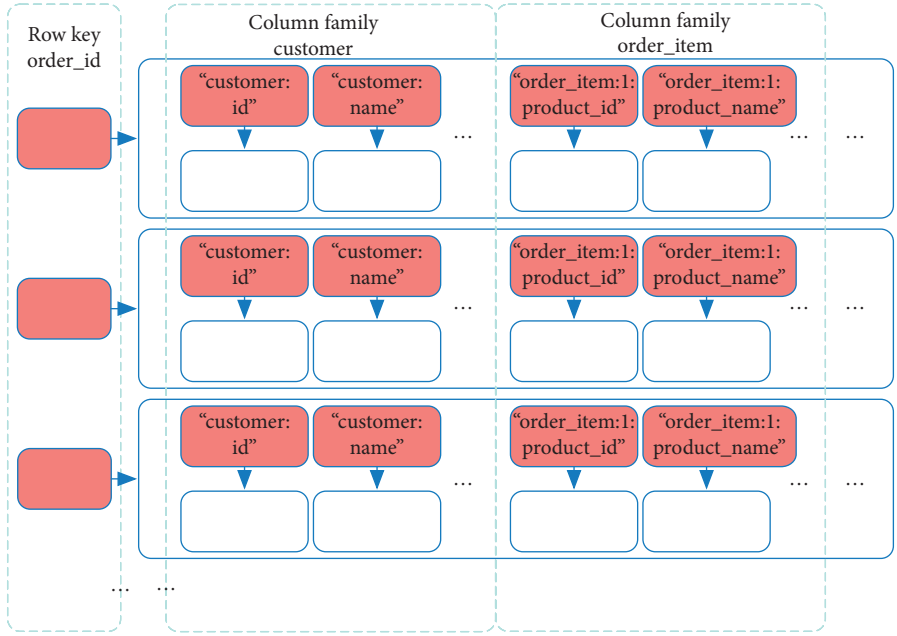

Figure 4: Column family model for E-shopping.

we also hope that they can maintain their logical connections (we will see how to achieve this in the subsequent discussion about the n-tier soft set model).

(iii) The form of key-value typed databases is changeful (known as schemaless databases), but it is not the case for query and reasoning (which is what the relational model good at). The contents of aggregates are prepared and stored for specific needs, and aggregates designed for an application are not necessarily suitable for others, which becomes another kind of inflexibility.

(iv) Key-value typed models have no rigorous mathematical basis. A strict mathematical foundation not only makes the definition and expression of the model more rigorous but also facilitates the theoretical study of the model, the deduction of its properties and theorems (or makes use of existing results), and the recognition of its logical reliability and completeness. It is also easy to design a concise and general query language (for example, the relational model achieves a powerful logical expression with a few operations).

(4) Graph. Graph models focus on solving the problem of lacking connections in the relational model and key-value typed models. As shown in Figure 6, the graph model consists of nodes and edges. Nodes are connected by edges, which can be directed or undirected. Nodes and edges can have attributes, which makes each look like a row in the column family model or a document in the document model. However, nodes are not separated but linked together by edges. In contrast, the main point of graph modeling is not to express the attributes of nodes or edges but to describe the connections between nodes. In our case, in the upper part of Figure 6, the followship network can be clearly expressed and easily queried by using a graph model, which is difficult to implement with the relational model and other NoSQL models. Based on graph theory, the graph model has a mature mathematical foundation and a large number of forthcoming achievements (theorems and algorithms), which makes it have the ability to deal with

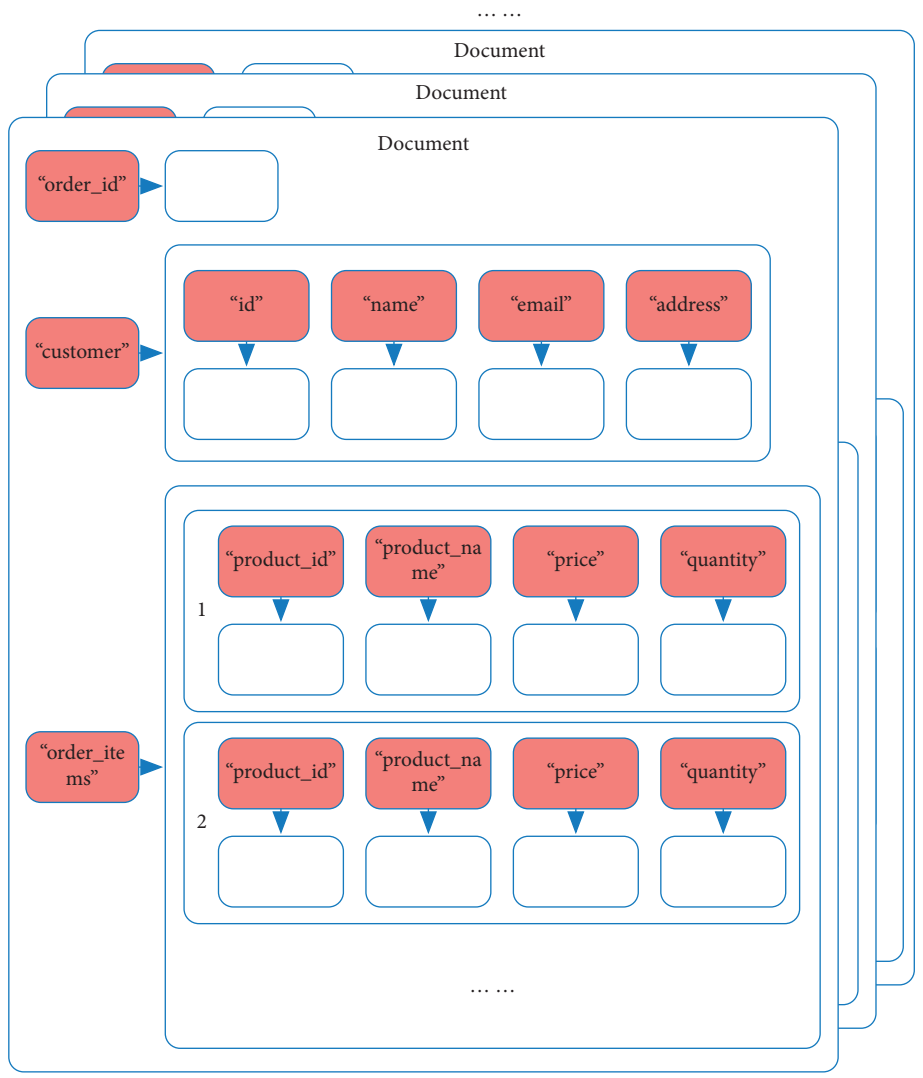

Figure 5: Document model for E-shopping.

connections easily and solve complex problems such as finding the shortest connection path between two nodes. However, when it comes to the issues that focus mainly on entities and their attributes (for example, classification or statistics reports), graph models have the same problems as other NoSQL models. For example, in order to count the proportion of male and female users in a followship network, we still need an external index to locate the nodes from attributes or count nodes by scanning the whole network.

3.2.3. Modeling with N-Tier Soft Set Model. Various models have been discussed above, as well as their problems. Now, let us take a look at how to modeling with the n-tier soft set model (hereafter referred to as the NTSS model).

(1) Rules for Modeling with the NTSS Model. Firstly, we introduce the rules for transforming the ER model into the NTSS model (other conceptual models can be deduced by the same way):

(i) Entities and attributes: as shown in Figure 7, entities and their attributes in the ER model are transformed into the connections of entity domains and attribute domains in the NTSS model. An entity domain is a set which is used to uniquely identify and represent entities. If the entities in the original ER model have simple artificial primary keys (such as IDs), they will be renamed (domain name in the NTSS model should be more descriptive) and converted into entity domains directly. If there are composite primary keys (composed of multiple attributes), simple artificial domains are added as entity domains.

(ii) Relations: as shown in Figure 8, relations which have no attribute in the ER model are represented by direct connections between entity domains in the NTSS model, and relations which have attributes are represented by connection domains and attribute domains connected with it. And if both sides of a relation are the same domain (such as self-relation),

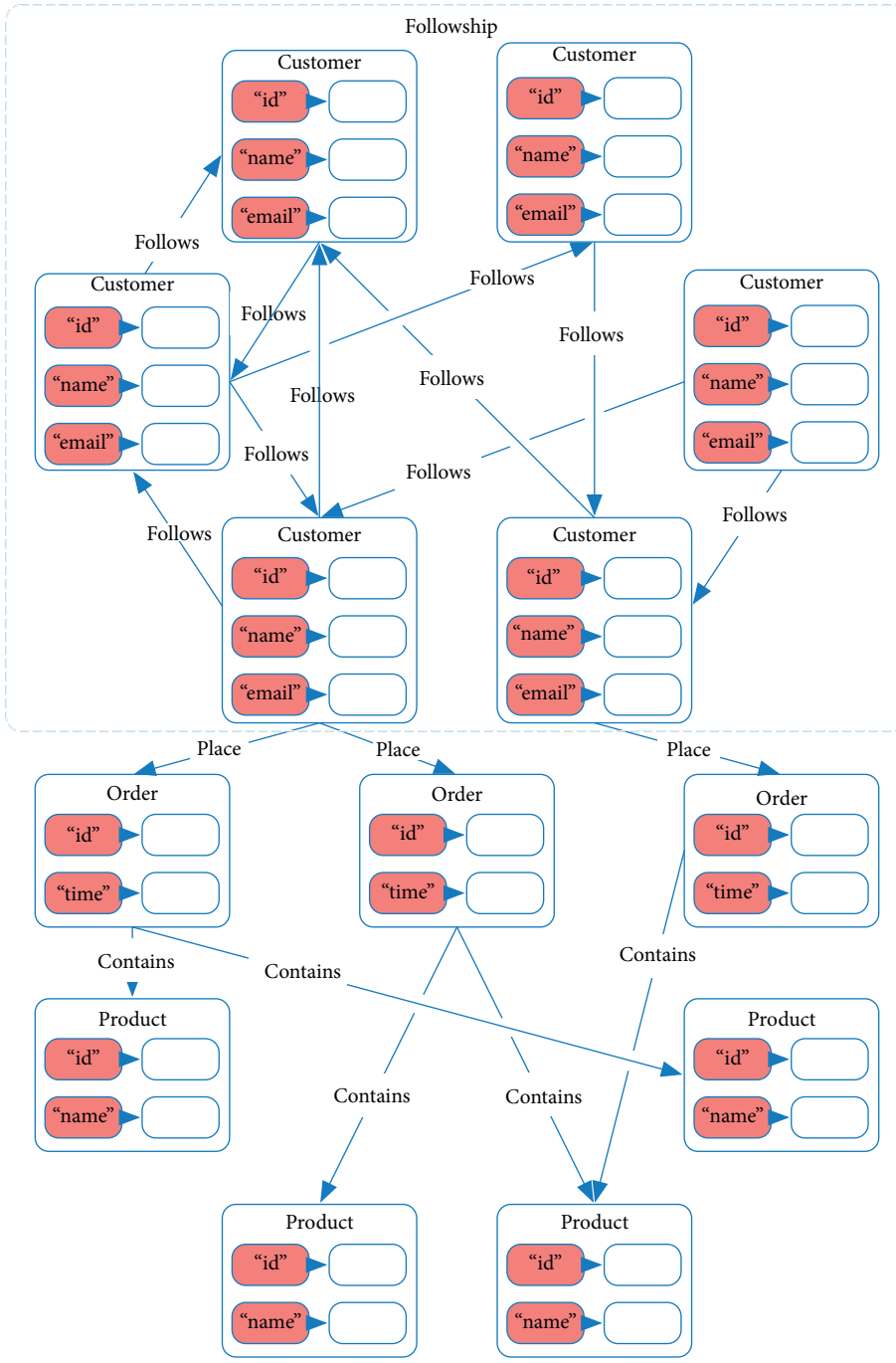

Figure 6: Graph model for E-shopping.

Figure 7: Entity in the NTSS model.

two role domains are added as a distinction for different roles.

(iii) Connections in the NTSS model: as shown in Figure 9, any connection in the NTSS model is

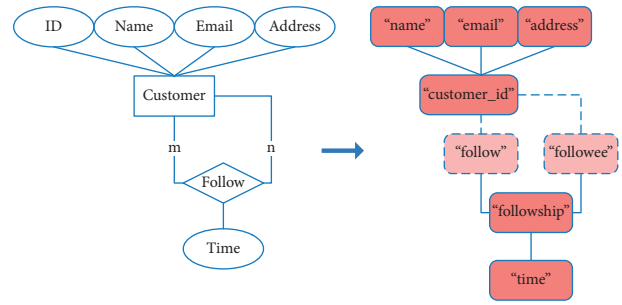

Figure 8: Relation in the NTSS model.

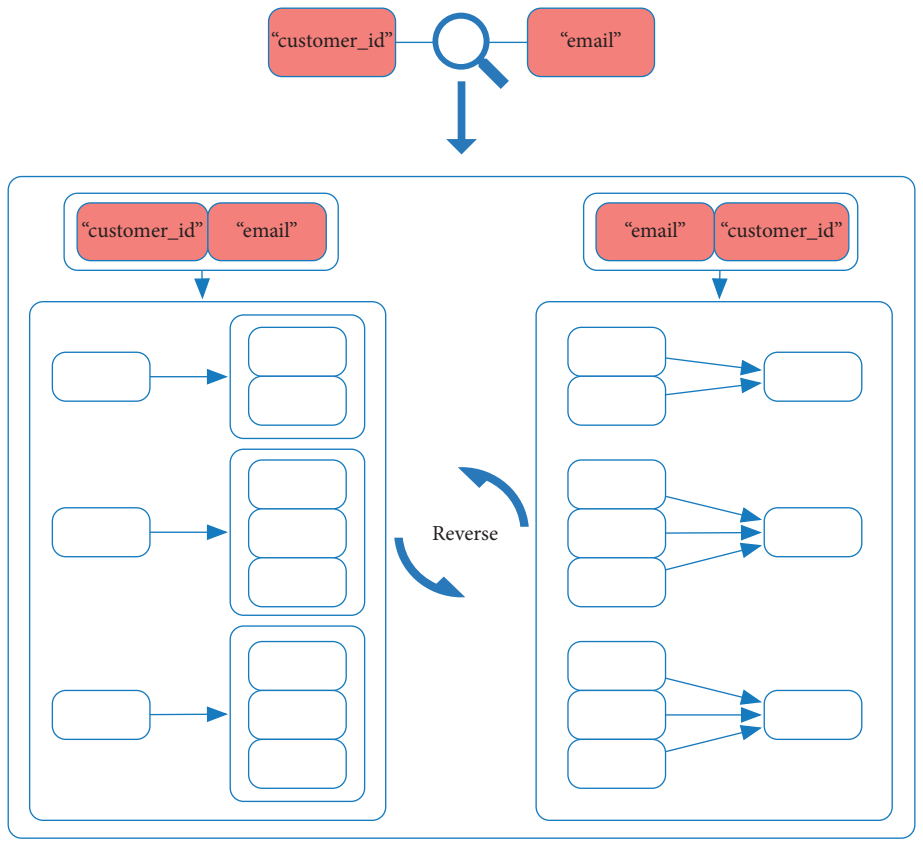

Figure 9: Connections in the NTSS model.

represented by a pair of domain relations whose names are reverse tuples (like ("customer_id," "e-mail") and ("e-mail, customer_id")) and values are reverse binary soft sets. The connection is undirected, and the data on both sides of the connection can be accessed symmetrically.

(iv) Cardinality constraint of a connection is expressed and implemented by the values of domain relation pairs. For any connection C between domain A and domain B, we have the following:

If it is a 1 : 1 connection, the values of domain relation pair C, $C(A, B)$: $A \longrightarrow P(B)$and $C(B, A)$: $B \longrightarrow P(A)$are both single-valued soft sets (all the images are either empty sets or just have only one element), and they are reverse of each other.
If it is a 1 : n connection, $C(A, B)$is a common soft set, $C(B, A)$is a single-valued soft set, and they are reverse of each other.
If it is an n : m connection, $C(A, B)$and $C(B, A)$are both common soft sets and they are reverse of each other.

For example, in Figure 9, the relation between "customer_id" and "e-mail" is a 1:n relation (one customer can have multiple e-mail addresses). So, the value of domain relation ("customer_id," "e-mail") is a common soft set, and the value of domain relation ("e-mail," "customer_id") is a single-valued soft set.

(2) Features and Advantages of the NTSS Model. The whole picture of converting the ER model in Figure 1to the NTSS model is shown in Figure 10.

(i) Macroscopically: we can see the similarities between the NTSS model and ER model in the upper half of the figure. The NTSS model, like conceptual models such as ER, retains the intuitive panorama of its modeling domain and constructs a network of domains, which has rich semantics and sufficient connections close to human natural thinking.

(ii) Microscopically: in the lower half of the figure, each connection between domains in the NTSS model is represented by a pair of named soft sets which are reverse of each other. An NTSS database is actually made up of such pairs of soft sets.

(iii) In implementation: an NTSS database is an n-tier soft set, so it can be uncurrying (Definition 22) as a multivariate function, which can be implemented by key-value pairs. For example, a piece of information about customer's names in an NTSS database "E–Shopping" can be represented as

{
"E–Shopping":{

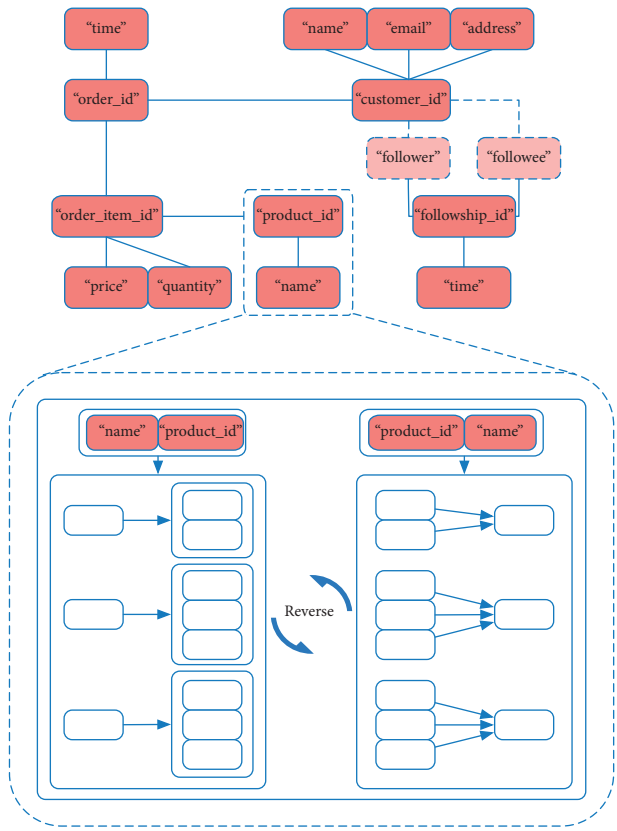

Figure 10: NTSS model for E-shopping.

("customer_id," "name"):{
"0001":"Joe",
"0002":"Eve",
...
},
("name," "customer_id"):{
"Joe":{"0001," "0086," "0223,"... },
"Eva":{"0002," "0332," "0487,"... },
...
}
}
}

which is a 4-tier soft set, and can be transformed into key-value pairs as

{
"E–Shopping, (customer_id, name), 0001":"Joe",
"E–Shopping, (customer_id, name), 0002":"Eva",
...
"E–Shopping, (name, customer_id), Joe":{"0001," "0086," "0223,"... },
"E–Shopping,(name, customer_id), Eva": {"0002," "0332," "0487,"... },
...
}

So, if we use a hashtable to be the underlying implementation of an NTSS database, the information contained in the keys will be implied in storage addresses, and values will be hashed but maintain the logical structure of the database.

In usage: through our formal definitions, for the upper application programming users, an NTSS database is just a function with a set of well-defined operations and uniform specifications. In fact, referring to the example mentioned above, let B be the database soft set which contains the "E–Shopping" database, and in upper programming languages, the database soft set B is just a function which return values are also functions. By giving a parameter "E–Shopping," B ("E–Shopping") returns the value (a domain relation soft set) of a database named "E–Shopping," which can still be regarded as a function. By giving a parameter ("customer_id," "name"), then B ("shopping") ("customer_id," "name") will return the value of the domain relation (still a function) between "customer_id" and "name." By giving a "customer_id" such as "0001," then B ("shopping") ("customer_id," "name") ("0001") will return the name of the customer. This is very natural to the language, which supports functional programming, and naturally constitutes a concise query language.

Next, we will expound the advantages of the NTSS model and explain why the NTSS model is suitable for dealing with big data.

First, we show the performance advantages of the NTSS database over the relational database through a comparative experiment. We implemented a prototype database based on NTSS (using Python) and compared it to 8.0.15 version of MySQL on a computer with 2.6 GHz Intel Core i7, 16 GB 1600 MHz DDR3, and 512 GB PCI SSD. We built three experimental data tables, Customer, Product, and Buy, to express the records of customers purchasing products. Each time 10,000 records of data are written, the time consumption of write is recorded, then the names of the customers who purchased the random 5 products are queried, and the time consumption of read is recorded.

MySQL write and read statements are similar to the following:

```
# Write
insert into customer (cust_id, cust_name, cust_sex) values (“c00001,” “Joe,” “male”)
insert into product (prod_id, prod_name, prod_desc) values (“p00001,” “tv,” “just_a_television”)
insert into buy (cust_id, prod_id, but_time) values (“c00001,” “p00001,” “20190101163749″)
# Read
select prod_name, cust_name from cust a, buy b, prod c where a.cust_id = b.cust_id and b.prod_id = c.prod_id and prod_name in (“tv,” “phone,” “pad,” “car,” “coke”)
```

NTSS write and read statements are similar to the following:

```
import ntssdb as nb
nb = nb.connect (host = “localhost,” dbname = “test,” user = “root,” password = “pw”)
# Write
nb (“cust_id,” “cust_name,” “cust_sex”).put (“c00001,” “Joe,” “male”)
nb (“prod_id,” “prod_name,” “prod_desc”).put (“p00001,” “tv,” “just a television”)
nb (“buy_id,”“cust_id,” “prod_id,” “buy_time”).put (“b00001,” “c00001,” “p00001,” “20190101163749”)
# Read
nb (“prod_name,” “prod_id,” “buy_id,” “cust_id,” “cust_name”).get (“tv,” “phone,” “pad,” “car,” “coke”)
```

In the experiment, we compared five key indicators with MySQL:

(1) When MySQL is not indexed, the insertion time increases with the amount of data.

(2) When MySQL is not indexed, the reading time increases with data.

(3) When MySQL is indexed, the insertion time increases with the amount of data,

(4) When MySQL is indexed, the reading time increases with the amount of data.

(5) Space usage.

It can be seen from Figures 11–13that when MySQL has no index, the insertion time can be regarded as the constant time of O(1). The random read time is O(n) (the whole table needs to be traversed), while the insertion and read time of NTSS are both the constant time of O(1) (the hash table is directly inserted and read). Still, the specific time consumed by each record during insertion is about four times slower than that of MySQL. However, due to the complexity of O(1), when reading, it is much faster than MySQL without an index.

When MySQL was indexed, the insertion and reading time is O(log(n)) in theory (because the index of MySQL is usually implemented by $B+$ tree), while NTSS is O(1). From the actual test data, we can see that the insertion and reading of MySQL increase with the increase of the amount of data, while the insertion and reading of NTSS fluctuate stably in a certain range.

For space usage, NTSS is about 2.73 times as large as a nonindexed MySQL database (NTSS: 402 MB, MySQL: 147 MB) to store the same data. However, if MySQL wants to query more freely (index all columns), its index space will be about 307 MB, so it will take up 147 + 307 = 454 MB in total, which is higher than that of NTSS.

We do not compare performance with the current NoSQL databases. As a prototype database implemented in Python, there is no comparability between NTSS and the mature NoSQL database that has evolved for many years in performance. Compared with the current NoSQL database, NTSS has the advantages of query freedom and mathematical logicality. Taking MongoDB as an example, as a popular database, MongoDB is widely applied in everyday applications and has extremely high performance in some queries, but it has no mathematical logicality and cannot query freely (strong at query key to value, but weak at query key to value). So, if you need to get the relationship between the values, it will cost a lot (need index structure or traverse scan). However, the NTSS model is a model with complete mathematical logicality and can query freely between key and value. The NTSS database cannot compete with MongoDB from an implementation perspective because NTSS only stays at the prototype level and will gradually approach the current mainstream NoSQL database through future improvements.

Based on the above experiments and previous discussions, we can clearly see that the NTSS model has the following advantages:

(i) Efficient performance: as we have seen in the comparison experiment, MySQL is a relational database whose data and indexes are separate, and its performance depends on the design of indexes; the write and read performance and the convenience of query cannot be taken into account at the same time. However, an NTSS database can be transformed to key-value pairs and implemented

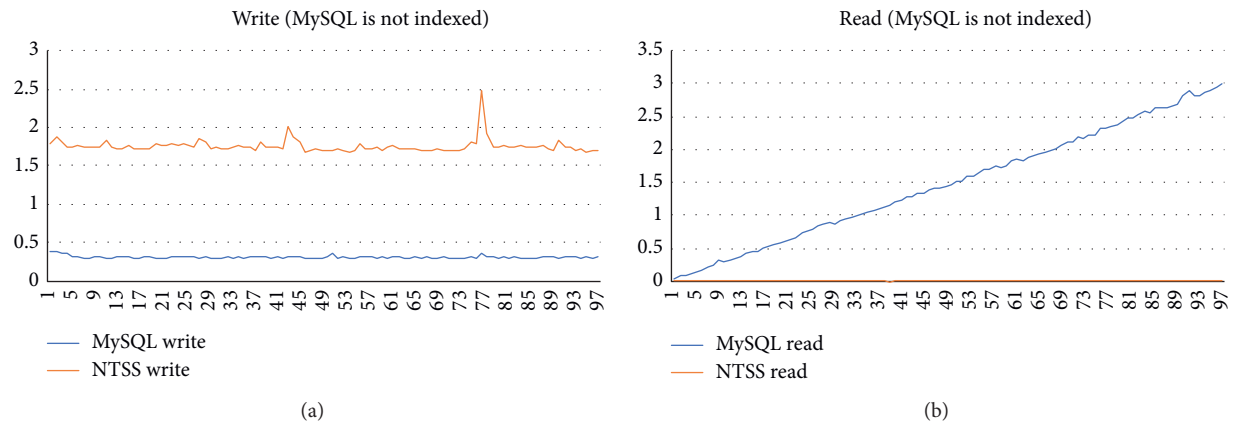

Figure 11: Performance comparison between NTSS and unindexed MySQL.

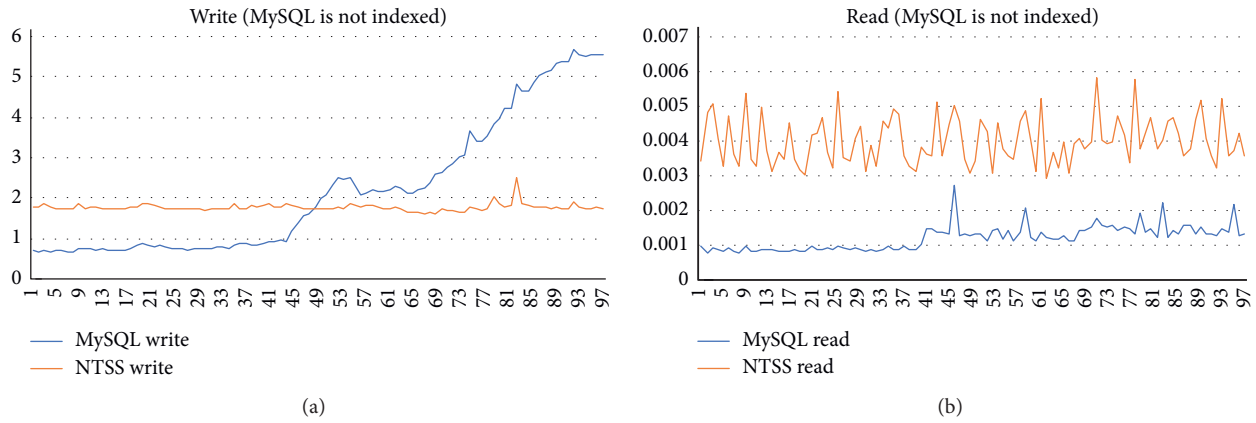

Figure 12: Performance comparison between NTSS and indexed MySQL.

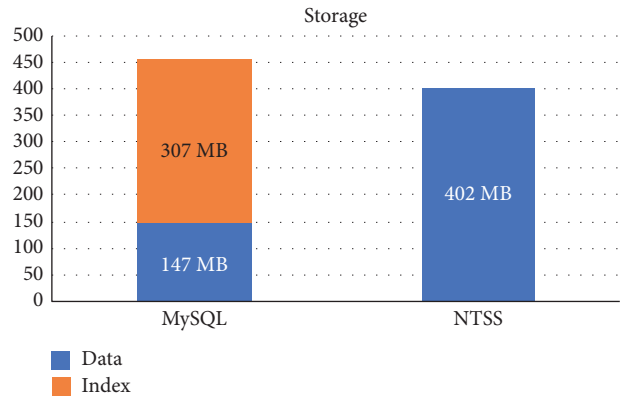

Figure 13: Comparison of storage consumption between MySQL and NTSS DB.

as a hashtable directly; therefore, any data in it can be write or read with an average time complexity of $O(1)$.

(ii) Schemaless: the NTSS model represents entities or aggregates as interconnections between domains, rather than a fixed table. Connections in the NTSS model are logically represented by n-tier soft sets and implemented by key-values in the underlying, which are independent of each other and can be added or deleted at will without mutual influence. This solves flat and rigid problems in the relational model. For example, if we want to split the "name" domain which is connected to "customer_id" into "first_name" and "last_name," we only need to add two new connections between "first_name," "last_name," and "customer_id" and delete the original one. This does not affect other parts of the database neither logically nor physically.

(iii) Semantic and data integration: the NTSS model represents semantics and data in an integrated way, which makes it is easier to move and disperse. It is no longer necessary to process metadata separately.

(iv) Index and data integration: an instance of the NTSS model is a nested index structure, and each atomic datum has a unique logical access path. The data stored in a database formed by the NTSS model are a complete index system itself, and every domain in it can be used as index key to indicate

Table 1: Comparison of data models.

	Relational	Key-value	Column family	Document	Graph	NTSS
Strict algebra basis	Yes	No	No	No	Yes	Yes
Completeness and consistency	Strong	Weak	Weak	Weak	Strong	Weak (but with native cardinality constraints)
Query expressiveness	Strong	Unidirectional	Multilevel unidirectional	Multilevel unidirectional	Strong	Strong
Data self-descriptive	No	Weak	Strong	Strong	Strong	Strong
Distributed support	Difficult	Easy	Easy	Easy	Difficult	Easy
Schema flexibility	Pre-defined rigid schema	Schemaless	Schemaless	Schemaless	Schemaless	Schemaless
Connecting data	Difficult	Difficult	Difficult	Difficult	Easy	Easy
Dependency on external indexes	Strong	Weak	Medium	Medium	Medium	No

data in other domains connected to it, which solves the problem of index and data separation of the relational model. And it becomes the key to efficient performance and sufficient connections.

(v) Sufficient connections: the atomic data in an NTSS database are no longer isolated, but in a network. In the NTSS model, entity domains are connected to each other, and attribute domains are connected to entity domains. These connections are static states of the model, and each connection is bidirectional. This solves the problems of lack of connection in the relational model or the key-value typed models, and the key-value typed model can only be accessed in one direction.

(vi) Rigorous mathematical foundation: based on n-tier soft set theory, the NTSS model has a rigorous formal definition. That is not available in other key-value typed models. This not only makes the NTSS model more precise in definition and expression but also facilitates more in-depth theoretical research. It enables us to infer richer properties (or to use the existing mathematical research results of soft sets) and to understand its logical reliability and completeness. It is convenient to design a concise and general query language and achieve complete logical expression ability with as few operations as the relational model.

(vii) Powerful query ability: through the rigorously defined operations, fast access brought by index and data integration, and sufficient connection between data, the NTSS model has the ability to query as freely and completely as the relational model but in a big data environments. In the comparison experiment with MySQL, we not only write and read key-values but also write the same logical structure as the relational model and implement the same query as the multi-table join SELECT SQL statement.

(viii) Convenient for programming usage: from a programming perspective, all the structures that make up the NTSS model include tuples, sets, and dictionaries are built into most programming languages and can be processed natively.

(ix) Easy to modeling: from the similarity between the NTSS model and the ER model, it can be seen that the macroscopic view of the NTSS model is close to the original appearance of human thinking and modeling, so that modeling can be carried out intuitively.

(x) Convenient for statistical use: each domain can be used as a statistical dimension, and most of the values related to it have become a set that can be directly obtained. For these sets: counts, sums, averages, and other statistical indicators are easy to calculate.

Using the conclusions in [3, 5–8], we summarize the difference between the relational model, the four NoSQL models, and the NTSS model as shown in Table 1.

Through the discussion above, we can see that the NTSS model is indeed a data model suitable for dealing big data with 4 Vs. For Volume, an NTSS database is a discrete key-value structure and has natural support for distributed clusters. For Velocity, the underlying implementation of key-values provides fast and flexible data processing. For Variety, as a schemaless model, it can be altered at will, making it easy to respond to changing requirements or different data sources. For Value, the complete logical structure is preserved between the data and can be queried freely, and storing set values also facilitates statistics and data mining. Moreover, based on the features of the NTSS model, it is possible to realize an implementation with intelligent data distribution, which can automatically adapt to the status of the cluster, intelligently divide the soft aggregations, and still maintain the semantic and logical structure between the data, without manual sharding design or aggregation design.

4. Conclusion

The n-tier soft set theory and n-tier soft set data model have been proposed. We defined them in a strict formalized way and illustrated the process and design considerations. We explained why and how to use the n-tier soft set model to modeling, described the features and advantages of it.

However, a lot of details have not been covered, such as richer algebraic properties and detailed implementation aspects, which will be progressively fulfilled in the future.

However, we believe that through this paper, we have not only expanded the frontier of soft set theory but also shed light on a promising prospect of developing a new database product based on the NTSS model to meet the challenge of big data. In the future, the database will be rewritten using Scala, unlike a theoretical verification based on Python Implementation currently and open-source to improve its ability.

Data Availability

The data used to support the findings of this study are available upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the National Natural Science Foundation of China (grant no. 72071021).

References

[1] J. Manyika, M. Chui, B. Brown et al., "Big data: the next frontier for innovation, competition, and productivity," Technical Report, McKinsey Global Institute, New York, NY, USA, 2011.

[2] J. Gantz and D. Reinsel, "The digital universe in 2020," Technical Report, IDC, Needham, MA, USA, 2012.

[3] M. Stonebraker, "SQL databases v. NoSQL databases," *Communications of the Acm*, vol. 53, no. 4, pp. 10-11, 2010.

[4] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[5] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM Sigmod Record*, vol. 39, no. 4, pp. 12–27, 2010.

[6] J. M. Patel, "Operational NoSQL systems: what's new and what's next?" *Computer*, vol. 49, no. 4, pp. 23–30, 2016.

[7] A. Corbellini, C. Mateos, A. Zunino, D. Godoy, and S. Schiaffino, "Persisting big-data: the NoSQL landscape," *Information Systems*, vol. 63, pp. 1–23, 2017.

[8] A. Davoudian, L. Chen, and M. Liu, "A survey on NoSQL stores," *ACM Computing Surveys*, vol. 51, no. 2, pp. 40:1–40: 43, 2018.

[9] P. Atzeni, F. Bugiotti, L. Cabibbo, and R. Torlone, "Data modeling in the NoSQL world," *Computer Standards & Interfaces*, vol. 67, p. 103149, 2020.

[10] M. V. Sokolova, F. J. Gómez, and L. N. Borisoglebskaya, "Migration from an SQL to a hybrid SQL/NoSQL data model," *Journal of Management Analytics*, vol. 7, no. 1, pp. 1–11, 2019.

[11] D. Molodtsov, "Soft set theory—first results," *Computers & Mathematics with Applications*, vol. 37, no. 4-5, pp. 19–31, 1999.

[12] M. Riaz, F. Karaaslan, I. Nawaz, and M. Sohail, "Soft multi-rough set topology with applications to multi-criteria decision-making problems," *Soft Computing*, vol. 25, no. 1, pp. 799–815, 2021.

[13] E. Aygün and H. Kamacı, "Some new algebraic structures of soft sets," *Soft Computing*, vol. 25, no. 13, pp. 8609–8626, 2021.

[14] F. Fatimah and J. C. R. Alcantud, "The multi-fuzzy N-soft set and its applications to decision-making," *Neural Computing and Applications*, vol. 33, pp. 1–10, 2021.

[15] J. Yang and Y. Yao, "Semantics of soft sets and three-way decision with soft sets," *Knowledge-Based Systems*, vol. 194, Article ID 105538, 2020.

[16] P. K. Maji, R. Biswas, and A. R. Roy, "Soft set theory," *Computers & Mathematics with Applications*, vol. 45, no. 4-5, pp. 555–562, 2003.

[17] C. F. Yang, "A note on "soft set theory"" *Computers & Mathematics with Applications*, vol. 56, no. 7, pp. 1899-1900, 2008.

[18] M. I. Ali, F. Feng, X. Liu, W. K. Min, and M. Shabir, "On some new operations in soft set theory," *Computers & Mathematics with Applications*, vol. 57, no. 9, pp. 1547–1553, 2009.

[19] P. Zhu and Q. Wen, "Operations on soft sets revisited," *Journal of Applied Mathematics*, vol. 2013, no. 5, 7 pages, Article ID 105752, 2013.

[20] A. Sezgin and A. O. Atagün, "On operations of soft sets," *Computers & Mathematics with Applications*, vol. 61, no. 5, pp. 1457–1467, 2011.

[21] K. Gong, Z. Xiao, and X. Zhang, "The bijective soft set with its operations," *Computers & Mathematics with Applications*, vol. 60, no. 8, pp. 2270–2278, 2010.

[22] X. Q. Zhou, Q. G. Li, and L. K. Guo, "On generalised interval-valued fuzzy soft sets," *Journal of Applied Mathematics*, vol. 2012, no. 11, 18 pages, Article ID 479783, 2012.

[23] Y. Jiang, Y. Tang, Q. Chen, H. Liu, and J. Tang, "Extending fuzzy soft sets with fuzzy description logics," *Knowledge-Based Systems*, vol. 24, no. 7, pp. 1096–1107, 2011.

[24] N. Çağman, S. Enginoğlu, and F. Citak, "Fuzzy soft set theory and its applications," *Iranian Journal of Fuzzy Systems*, vol. 8, no. 3, pp. 137–147, 2011.

[25] S. Alkhazaleh, A. R. Salleh, and N. Hassan, "Possibility fuzzy soft set," *Advances in Decision Sciences*, vol. 2011, no. 3, 18 pages, 2011.

[26] P. Majumdar and S. K. Samanta, "Generalised fuzzy soft sets," *Computers & Mathematics with Applications*, vol. 59, no. 4, pp. 1425–1432, 2010.

[27] A. R. Roy and P. K. Maji, "A fuzzy soft set theoretic approach to decision making problems," *Journal of Computational and Applied Mathematics*, vol. 203, no. 2, pp. 412–418, 2007.

[28] B. Qin, "Topological structures of soft fuzzy rough sets," *Journal of Computational Analysis and Applications*, vol. 17, no. 3, pp. 459–467, 2014.

[29] Z. Li and T. Xie, "The relationship among soft sets, soft rough sets and topologies," *Soft Computing*, vol. 18, no. 4, pp. 717–728, 2014.

[30] C. F. Yang, "Soft rough sets and their properties," *Journal of Computational Analysis and Applications*, vol. 15, no. 7, pp. 1291–1299, 2013.

[31] F. Feng, X. Liu, V. Leoreanu-Fotea, and Y. B. Jun, "Soft sets and soft rough sets," *Information Sciences*, vol. 181, no. 6, pp. 1125–1137, 2011.

[32] A. O. Aygün and E. Aygun, "Groups of soft sets," *Journal of Intelligent & Fuzzy Systems*, vol. 30, no. 2, pp. 729–733, 2016.

[33] M. I. Ali, M. Shabir, and M. Naz, "Algebraic structures of soft sets associated with new operations," *Computers & Mathematics with Applications*, vol. 61, no. 9, pp. 2647–2654, 2011.

[34] U. Acar, F. Koyuncu, and B. Tanay, "Soft sets and soft rings," *Computers & Mathematics with Applications*, vol. 59, no. 11, pp. 3458–3463, 2010.

[35] F. Feng, Y. B. Jun, and X. Zhao, "Soft semirings," *Computers & Mathematics with Applications*, vol. 56, no. 10, pp. 2621–2628, 2008.
[36] H. Aktas and N. Çağman, "Soft sets and soft groups," *Information Sciences*, vol. 177, no. 13, pp. 2726–2735, 2007.
[37] N. Çağman and S. Enginoğlu, "Soft set theory and uni–int decision making," *European Journal of Operational Research*, vol. 207, no. 2, pp. 848–855, 2010.
[38] S. Danjuma, T. Herawan, M. A. Ismail, H. Chiroma, A. I. Abubakar, and A. M. Zeki, "A review on soft set-based parameter reduction and decision making," *IEEE Access*, vol. 5, pp. 4671–4689, 2017.
[39] X. Q. Ma and H. W. Qin, "A distance-based parameter reduction algorithm of fuzzy soft sets," *IEEE Access*, vol. 6, pp. 10 530–610 539, 2018.
[40] S. Danjuma, M. A. Ismail, and T. Herawan, "An alternative approach to normal parameter reduction algorithm for soft set theory," *IEEE Access*, vol. 5, pp. 4732–4746, 2017.
[41] Z. W. Li and N. H. Gao, "The parameter reduction of soft sets and its algorithm," *Journal of Computational Analysis and Applications*, vol. 16, no. 1, pp. 76–84, 2014.
[42] X. Ma, N. Sulaiman, H. Qin, T. Herawan, and J. M. Zain, "A new efficient normal parameter reduction algorithm of soft sets," *Computers & Mathematics with Applications*, vol. 62, no. 2, pp. 588–598, Jul. 2011.
[43] Z. Kong, L. Gao, L. Wang, and S. Li, "The normal parameter reduction of soft sets and its algorithm," *Computers & Mathematics with Applications*, vol. 56, no. 12, pp. 3029–3037, 2008.
[44] L. Li, Z. Xiao, X. D. Feng, and B. Zhong, "Soft incomplete discernibility matrix for decision-making problems," *IEEE Access*, vol. 6, pp. 32 450–32 459, 2018.
[45] F. Y. Xiao, "A hybrid fuzzy soft sets decision making method in medical diagnosis," *IEEE Access*, vol. 6, pp. 25 300–25 312, 2018.
[46] H. Aktas and N. Çağman, "Soft decision making methods based on fuzzy sets and soft sets," *Journal of Intelligent & Fuzzy Systems*, vol. 30, no. 5, pp. 2797–2803, 2016.
[47] F. Chang, J. Dean, S. Ghemawat et al., "Bigtable: a distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.