A Language and a Physical Organization Technique for Summary Tables+

Gultekin Ozsoyoglu, Z Meral Ozsoyoglu and Francisco Mata#

Department of Computer Engineering and Science
and
Center for Automation and Intelligent Systems
Case Western Reserve University
Cleveland, Ohio 44106

ABSTRACT

A summary table is a tabular representation of summary data, and is a useful data structure for statistical databases Primitive summary tables are basic building blocks of summary tables, and can be represented as relations with set-valued attributes In this paper, we propose a set of summary table manipulation operators that, together with an algebra of set-valued relations, form an algebraic language for manipulating set-valued relations and arbitrary summary tables We then describe a physical organization technique for summary tables, and discuss an implementation for summary table operators utilizing this technique

1. Introduction

Databases that are mainly used for statistical analysis are called statistical databases (SDB) A statistical database management system (SDBMS) may be defined as a database management system that provides capabilities (i) to model, store, and manipulate data, and (ii) to apply statistical data analysis techniques to data in the database The special utilization characteristics of SDBs necessitate incorporation of new objects into data models for SDBs, such as histograms, cross-tabulations, scatter diagrams [Shos82, McCa82, Olke82, SuNB82], and new database languages using these objects

Tabular representations of summary data, called summary tables, are widely used in various SDB application areas such as management decision making, health care, economic planning, and census data evaluation [OzsO83] Fig 1 shows an instance of a summary table

1980-COUNT				*AGE {11, ,40}	*AGE {41, ,70}
COUNTRY	USA	SEX.	M	115	70
			F	85	60
		GAME	Alpine-skiing	18	3
	CAN	SEX	M	75	15
			F	80	20
		GAME	Nordic-skiing	20	15

Figure 1 An Instance of the Summary Table 1980-COUNT

The use of summary tables is not restricted to output formatting, they are maintained for bookkeeping, compared and evaluated perhaps over a time span Assuming that summary tables are proper logical modeling tools, we need a data manipulation language for summary tables Moreover, when a database contains a large

+ This research is supported by the National Science Foundation under grant MCS-8306616
# The work of F Mata is supported by the Organization of American States

number of summary tables, it is also crucial to have efficient storage techniques for them

Statistical packages usually have commands to produce summary tables (e g TABLES command of P-STAT [Psta81], and BREAKDOWN and CROSSTABS procedures of SPSS [Nie75]) However, these commands have restricted summary table definition and manipulation capabilities For example, they create only special classes of summary tables (that we call primitive summary tables), and from a single file Moreover the syntax of these commands is usually complex and difficult to follow Another system that is exclusively designed to produce summary tables is the table Producing Language (TPL) of the U S Bureau of Labor Statistics[Uslb80] TPL has the power to produce arbitrary summary tables, and is extensively used However, it does not manipulate summary tables, and is a stand-alone system used in batch mode to extract and tabulate data from a single file Ikeda and Kobayashi [IkeK81] describe an implementation which extends Model 204 Database Management System [CCA79] with an operator to create summary tables, and three operators for manipulation of summary tables

Previously we have proposed a relational calculus-based language, called the Summary-Table-by-Example (STBE) [OzsO84] that manipulates summary tables and relations with set-valued attributes (*set-valued relations*) This paper proposes an algebraic counterpart to STBE -- a set of operators to manipulate summary tables This set of operators, together with an algebra of set-valued relations, have the capability to express arbitrary queries involving summary tables and relations, such as manipulating summary tables, creating summary tables from a set of relations, transforming a summary table into a set of relations, etc We then propose a physical organization technique for summary tables, and briefly describe how the summary table operators are implemented using this technique

2. The Data Model

A database is a set of relation and summary table instances A *relation instance* (or, a relation) is a table with each column labeled by a distinct attribute Each attribute has an associated domain of values A *relation scheme* is a set of attributes We will assume that the attributes in a scheme are ordered, and thus refer to the i^{th} attribute in a relation scheme by integer i An attribute may be elementary (atomic) or set-valued An atomic attribute is an attribute whose values are elementary (numeric, character strings, etc) A set-valued attribute (prefixed by a '*') is one whose values are sets of elementary values

Informally, a summary table scheme is a two-dimensional array of cells The rows and columns of a summary table are labeled by attributes, called *category attributes*, which may be structured in the form of an ordered set of trees, called *row or column category attribute forests* Each cell in a summary table has an associated set of row or column category attributes which serve as row and column indices for the cell The set of row (column) category attributes of a cell forms a path from the root to a leaf in a row (column) category attribute tree Each cell in a summary table is also labeled by an attribute, called *cell attribute*

Example 1. Consider the summary table scheme in fig 2 a This summary table has two cells with cell attributes $COUNT_1$ and $COUNT_2$ The row category attribute forest consists of one tree whose root is COUNTRY, and the leaves are SEX and GAME The column category attribute forest consists of a tree with one node, which is *AGE

1980-COUNT		*AGE
COUNTRY	SEX	$COUNT_1$
	GAME	$COUNT_2$

(a) Scheme of the summary table in Example 1

S_1			*AGE {11, ,40}	*AGE {41, ,70}
COUNTRY	USA	SEX M	115	70
		SEX F	85	60
	CAN	SEX M	75	15
		SEX F	80	20

S_2			*AGE {11, ,40}	*AGE {41, ,70}
COUNTRY	USA	GAME Alpine-skiing	18	3
	CAN	GAME Nordic-skiing	20	15

(b) Primitive Summary Tables of Example 1

R_1

COUNTRY	SEX	*AGE	$COUNT_1$
USA	M	{11, ,40}	115
USA	M	{41, ,70}	70
USA	F	{11, ,40}	85
USA	F	{41, ,70}	60
CAN	M	{11, ,40}	75
CAN	M	{41, ,70}	15
CAN	F	{11, ,40}	80
CAN	F	{41, ,70}	20

R_2

COUNTRY	GAME	*AGE	$COUNT_2$
USA	Alpine-skiing	{11, ,40}	18
USA	Alpine-skiing	{41, ,70}	3
CAN	Nordic-skiing	{11, ,40}	20
CAN	Nordic-skiing	{41, ,70}	5

(c) Set-Valued Relations R_1 and R_2 that are Information-Equivalent to S_1 and S_2

Figure 2 Summary Tables and Set-Valued Relations

More formally, a *summary table scheme* S(Fr,Fc,C) is a three-tuple where Fr and Fc denote row and column category attribute forests, and C is an ordered multi-set of cell attributes We utilize a parenthesized expression to specify a category attribute tree which is a preorder enumeration of the tree (i e , first the root then the subtrees from left to right) In example 1, the expression for Fr is COUNTRY(SEX,GAME) and the summary table is 1980-COUNT ((COUNTRY(SEX,GAME)), (*AGE), (COUNT1, COUNT2)) Since the category attribute forests are ordered sets of trees each tree in Fr (Fc) is referenced by its position in the ordering in Fr (Fc) In a summary table, a category attribute may be elementary or set-valued, but the cell attributes are always elementary

A summary table with exactly one cell is called a *primitive summary table* It follows that if S(Fr,Fc,C) is a primitive summary table then $|Fr| = |Fc| = |C| = 1$, and each tree in Fr and Fc has exactly one leaf The summary table in Example 1 has two primitive summary tables as shown in Fig 2 b Since a summary table is a collection of cells where each cell corresponds to a primitive summary table, primitive summary tables are considered as building blocks of summary tables

A *summary table instance* is a collection of cell instances structured as specified by the summary table scheme A cell instance consists of values of its row and column category attributes and a value for its cell attribute Fig 1 shows an instance of the summary table 1980-COUNT whose scheme is given in Fig 2 a Note that, in 1980-COUNT, there are eight instances for the cell with cell attribute $COUNT_1$, and four instances for the cell with attribute $COUNT_2$ Instances of the primitive summary tables S1 and S2 corresponding to 1980-COUNT are shown in fig 2 b

From now on, when there is no confusion, we will use the term 'summary table' to denote either the scheme or the instance of a summary table Similarly we use the term 'cell' to denote

a cell attribute value and the associated category attribute values

2.1. Information Equivalence of Primitive Summary Tables and Set-Valued Relations

A relation possibly with set-valued attributes can be used to represent a primitive summary table excluding the order and the type (i e , row or column) of category attributes We now give the properties of such a relation in terms of functional dependencies and embedded join dependencies

Let V and W be subsets of attributes in a relation scheme R The *functional dependency* V $\rightarrow$ W holds in R if for any instance r of R, r has no two tuples which agree on V components, but disagree on one or more W components

Let R_1, ,R_k be nonempty subsets of the attributes in R such that $R_1 \cup \cup R_k = R$ A *join dependency* $*(R_1, ,R_k)$ is satisfied by a relation r over R if and only if the natural join of the projections of R onto R_i's equals R If $R_1 \cup \cup R_k = V$ where $V \subseteq R$, and a join dependency $*(R_1, , R_k)$ is satisfied by the projection of r for R onto V then $*(R_1, ,R_k)$ is called an *embedded join dependency* in r

Let X and Y be the disjoint sets of attributes in row- and column-attribute forests, respectively, of a primitive summary table S with the cell attribute C Then a relation R with the attributes X $\cup$ Y $\cup$ {C} can be used to represent S such that each tuple t in R corresponds to a cell value in S whose row and column category attribute values are tuple components in t[X] and t[Y], respectively, and whose cell attribute value is t[C] Let R be such a relation representing a primitive summary table S Then the functional dependency XY $\rightarrow$ C holds in R since each cell value in S is uniquely identified by category attribute values Moreover, the embedded join dependency *(X,Y) also holds in R since there is a cell value in S for every pair of X, Y values If R is such a relation representing a primitive summary table S, we say that R and S are *information equivalent* Similarly given an instance for R(X,Y,C) satisfying XY $\rightarrow$ C and *(X,Y) and a scheme for the corresponding primitive summary table S, an instance for S can be directly constructed

3. Summary Table Operations

The summary table manipulation language is designed by first defining operations to construct and manipulate primitive summary tables, and then by extending the language to deal with arbitrary summary tables This approach is motivated by the information equivalence of primitive summary tables and set-valued relations In this language, with the exception of two operators, operands and the result of the operation can be either relations or summary tables, bot not both This provides a clean separation of relation manipulation and summary table manipulation operators The exceptions are relation formation and primitive summary table formation operators (discussed later) which form in a sense an interface between summary table manipulation and relation manipulation parts of the language

Section 3 1 describes operations to create arbitrary summary tables Section 3 2 lists some redundant, but useful, operations To illustrate the operations, we use a Winter Olympics database with two relation schemes (about 1984 olympics)

ATHLETE (*NAME*,COUNTRY,SEX,
AGE,*EVENTS)

GAMES (*CLASS*,*EVENTS)

and two summary table schemes (about 1976 and 1980 Olympics)

1976-COUNT(Fr_1,Fc_1,($CT1$,$CT2$))

1980-COUNT(Fr_1,Fc_1,($COUNT1$,$COUNT2$))

where Fr_1=$COUNTRY(SEX,GAME)$ and Fc_1=$*AGE$ For relation schemes, the attributes in italic are keys, and *AGE and *EVENTS are

set-valued attributes Attribute CLASS refers to game classes (e g , Alpine-skiing or jumping) and EVENT refers to specific branches of a game class (e g , slalom skiing)

3.1. Basic Operations

Primitive Summary Table Formation (ST)

Let R be a (perhaps set-valued) relation, Atr(R) = X ∪ Y ∪ {C} where Atr(R) denotes the attributes of R, X and Y are disjoint sets, C is an atomic attribute and the functional dependency XY → C holds in R Then

$$ST_{(X,Y,C)}(R)$$

produces a primitive summary table where Fr and Fc each consists of a single tree with a single root-to-leaf path, X and Y, respectively

The primitive summary table formation operation creates instances for TR and TC and the two dimensional array of cell values If the embedded join dependency *(X,Y) does not hold in R then the constructed summary table will not be information equivalent to R, and therefore the array of cell values will have some cells with no cell attribute values To each such cell, we assign a *null value* ('-'), which stands for 'nonexistent'

Relation Formation (REL)

Let S be a primitive summary table where X and Y are row and column category attributes, C is the cell attribute, and X and Y are disjoint sets Then

$$REL(S)$$

produces a relation R where Atr(R) = X ∪ Y ∪ {C}, and for each cell in S, there is a tuple t in R such that t[X] and t[Y] are the same as the row and the column category attribute values of the cell and t[C] is the cell attribute value of the cell The mapping between cells in S and tuples in R is one-to-one and onto That is, if R = REL(S) then the functional dependency XY → C and the embedded join dependency *(X,Y) hold in R

Relation formation is an inverse of the primitive summary table formation operation in the following sense Let R(X,Y,C) be a relation, X and Y are disjoint sets, C is an atomic attribute Let S be a primitive summary table with the set of nodes in Fr and Fc being equal to X and Y respectively, and C is the cell attribute Then

(a) S = $ST_{(X,Y,C)}(REL(S))$, and

(b) R = REL($ST_{(X,Y,C)}(R)$) if the functional dependency XY → C and the embedded join dependency *(X,Y) hold in R

As an example, consider the primitive summary table S_1 in Fig 2 b The relation resulting from REL(S_1) is the relation R_1 shown in Fig 2 c

Concatenation of Two Summary Tables (CONC)

Let summary tables S_1 and S_2 have the same row category attribute forest Fr and its instance, S_1 and S_2 have column category attribute forests Fc_1 and Fc_2, respectively, and cell attribute sets A_1 and A_2 respectively Column concatenation of summary tables S_1 and S_2

$$CONC_C(S_1,S_2)$$

produces a summary table S with row category attribute forest Fr, column category attribute forest Fc = $Fc_1 \cup Fc_2$ (where ∪ denotes concatenation of ordered sets), and an ordered cell attribute set $A_1 \cup A_2$ such that for each cell x in S, if x is in S_1 then the cell attribute of x is in A_1, otherwise it is in A_2 The subscript can only be R or C denoting row or column concatenation, respectively

Example 2. Assume Fr instances of tables 1976-COUNT and 1980-COUNT are the same Then the operation

$CONC_C$ (1976-COUNT,1980-COUNT)

creates a new summary table whose scheme is as shown below

1976-1980-COUNT		*AGE	*AGE
COUNTRY	SEX	CT_1	$COUNT_1$
	GAME	CT_2	$COUNT_2$

Extract Summary Table (EX)

Let RT and ST be sets of integers denoting sets of trees in Fr and Fc of a summary table S, and let Ac be the ordered multiset of cell attributes of S Then

$$EX_{(RT,ST)}(S)$$

produces a summary table whose row and column category attribute forests are the trees of Fr and Fc as specified by RT and ST respectively, and whose cell attributes are those attributes in Ac corresponding to new category attribute forests

The extract summary table operation and the concatenate summary table operation are inverses of each other in the following sense Let S_1 and S_2 have the same row category attribute forest with a single tree which has the same instance in both S_1 and in S_2 Suppose the column category attribute forests of S_1 and S_2 each also has a single tree Then

$$S_1 = EX_{(\{1\},\{1\})}(CONC_C(S_1,S_2)), \text{ and}$$

$$S_2 = EX_{(\{1\},\{2\})}(CONC_C(S_1,S_2))$$

Attribute Split (SPLIT)

Attribute split and merge operations provide primitive/non-primitive summary table transformation capabilities Similarly, relation formation and primitive summary table formation provide relation/primitive summary table transformation capabilities Therefore a non-primitive summary table can be transformed into a set of (perhaps set-valued) relations and manipulated using relational algebra operators [OzsO83]

Let T with root A be a subtree of TC in a summary table S Assume A has k, k > 1, immediate descendants Let P denote the path of attributes from the root of TC to A The column split operation

$$SPLIT_{(C,TC,P)}(S)$$

maps T into a tree in which A is replaced by k new attributes, each named A, and each having exactly one descendant of the splitted attribute A as its child, in the original order The first subscript can only be R(ow) or C(olumn) specifying TC $\in$ Fr or TC $\in$ Fc, respectively This operation does not eliminate any cell values, it only relocates columns of the summary table

Example 3. Consider the summary table 1980-COUNT with the instance in given in fig 1 The operation

$SPLIT_{(R,1,(COUNTRY))}$ (1980-COUNT)

produces the summary table instance

NEW-1980-COUNT				*AGE {11, ,40}	*AGE {41, ,70}
COUNTRY	USA	SEX	M	115	70
			F	85	60
	CAN	SEX	M	75	15
			F	80	20
COUNTRY	USA	GAME	Alpine-skiing	18	3
	CAN	GAME	Nordic-skiing	20	15

Attribute Merge (MERGE)

Consider the column merge operation Let T be a tree of Fc in a summary table S where A with immediate successors B_1 and B_2 is a node in T If values of B_1 and B_2 are the same in a given summary table then the column-merge operation

$$MERGE_{(C,T,P,B_1,B_2)}(S)$$

merges B_2 to B_1, and the subtrees with root B_2 become subtrees of B_1 The third subscript P denotes the path from the root of T to A The first subscript can be R(ow) or C(olumn) specifying T $\in$ Fr or T $\in$ Fc, respectively When T and P are not specified then B_1 and B_2 are

root attributes of two distinct trees (specified by integers)

Example 4. Consider the summary table NEW-1980-COUNT obtained after applying the split operation to 1980-COUNT in example 3 The operation

$$MERGE_{(R,,,1,2)}\ (\text{NEW-1980-COUNT})$$

produces a summary table identical to 1980-COUNT

3.2. Other operations

Although operations in section 3 1 are powerful to manipulate arbitrary summary tables, expressions for some common summary table manipulation queries become quite long Below we introduce operations (expressible by basic operations) that simplify common expressions significantly

Aggregation-over-Table (AGT)

The operation

$$AGT_{(R,f)}\ (\text{S})$$

appends to summary table S a new column with an *empty tree* in Fc An empty tree has no attributes, however, it has a position in Fc Each cell attribute value c of the new column is obtained by applying the aggregate function f to the set of cell attribute values in the same row with c The first subscript can only be R or C indicating aggregation over rows or columns

Example 5. The operation

$$AGT_{(R,SUM)}\ (\text{1980-COUNT})$$

adds to 1980-COUNT a new column containing the total count of athletes for each (COUNTRY,SEX) instance and each (COUNTRY,GAME) instance The resulting summary table 1980-SUM-COUNT is shown below

1980-SUM-COUNT			*AGE {11, ,40}	*AGE {41, ,70}	
COUNTRY: USA	SEX:	M	115	70	185
		F	85	60	145
	GAME:	Alpine-skiing	18	3	21
COUNTRY: CAN	SEX:	M	75	15	90
		F	80	20	100
	GAME:	Nordic-skiing	20	15	35

Attribute Removal by Aggregation (AREM)

This operation deletes a leaf category attribute A from Fr or Fc by aggregating cell values distinguished by A Let T be a tree in Fc of summary table S, L be a leaf node in T, and P denote the path of attributes from the root of T to L Let N denote a set of cell attribute values that are qualified by attributes in P and, except L, by identical category attribute values The operation

$$AREM_{(C,T,P,f)}\ (\text{S})$$

removes L from T (i e makes each immediate ascendant of L a new leaf node), and for each N, computes a new cell attribute value by applying f to N

Example 6. The operation

$$AREM_{(C,1,*AGE,SUM)}\ (\text{1980-COUNT})$$

replaces all *AGE columns with a single column in which each value is the sum of all the values in that row of 1980-COUNT Since *AGE does not have a parent node the new column is qualified by an empty tree in Fc

Summary Table Formation from Several Primitive Summary Tables (TABLE)

Given Fc, Fr and primitive summary tables of a summary table S, this operation constructs an instance of S

Let Fr and Fc be two ordered forests Let $\{S_i \mid 1 \leq i \leq n\}$=K be an ordered set of primitive summary tables We assume that for each root-to-leaf path X in Fr and each root-to-leaf path Y in Fc there is a unique S_i in K such that X forms the row category forest of S_i and Y forms the column category forest of S_i The operation

$$TABLE_{(Fr,Fc)}(S_1,S_2, \ ,S_n)$$

forms a summary table S with Fr and Fc as its row and column category forests and K as the set of primitive summary tables of S Since for any X and Y, the corresponding S_i in K is uniquely defined, the cell attribute list for S and the instance of S can be unambiguously constructed

Example 7. Consider the primitive summary tables S_1 and S_2 in fig 2 b Let F1 and F2 denote the forests (COUNTRY(SEX,GAME)) and (*AGE) (we use a declarative statement to assign names to trees and forests) Then the operation $TABLE_{(F1,F2)}(S_1,S_2)$ creates a table identical to 1980-COUNT in fig 1

Decomposing a Summary Table into its Primitive Summary Tables (DEC)

In a given summary table scheme S, the row-by-row enumeration of cells defines an ordering M among the primitive summary tables of S For S composed of n primitive summary tables, the i^{th} primitive summary table refers to the primitive summary table at the i^{th} position in M The operation

$$DEC_I \ (S)$$

returns the I^{th} primitive summary table of S

Example 8. Consider the summary table 1980-COUNT instance in fig 1 The operation $DEC_2(1980\text{-}COUNT)$ extracts the summary table S_2 in fig 2 b

4. Algebra of Set-Valued Relations

This section summarizes the relational algebra of set-valued relations The algebraic operators include cartesian product, project, select, natural join, set union, set difference, set intersection, pack, unpack, set formation, aggregation-by-template and construct The cartesian product ($\times$), and project (Π) apply directly to set-valued relations For the set union ($\cup$), the set intersection ($\cap$) and the set difference (-), the corresponding attributes in both relations must be of the same type (i e simple- or set-valued) Selection (σ) and natural join of two relations [Ullm82] are extended to set-valued relations with minor modifications Below we discuss the remaining operators

Let t_1 and t_2 be two tuples having components for a set of attributes X Then $t_1[X]=t_2[X]$ denotes $t_1[X_i]=t_2[X_i]$ for all $X_i \in$ X D denotes the set of relations in the database and U is the set of elementary values

Pack (P)

Let R $\in$ D, | Atr(R) |= n, A $\in$ Atr(R) and $C_A = Atr(R) - \{A\}$ For each (n-1)-tuple $g \in \Pi_{C_A}(R)$, we define

$$W_g[C_A] = g$$

$$W_g[A] = \begin{cases} \{t[A] \mid t \in R \text{ and } t[C_A]=g\} \\ \quad \textit{if A is simple-valued} \\ \{x \mid (\exists t)(t \in R \text{ and } \\ \quad t[C_A]=g \text{ and } x \in t[A])\} \\ \quad \textit{otherwise} \end{cases}$$

$$\textit{then} \qquad P_A(R) = \{W_g \mid g \in \Pi_{C_A}(R)\}$$

The pack operator $P_A(R)$ maps (packs) sets of tuples in R, whose (n-1) components for C_A are the same, into a single tuple

Example 9. Consider the relation R_1 in fig 2 c The relations R_3 and R_4 where $R_3=P_1(\Pi_{1,2,3}(R_1)$ and $R_4=P_2(R_3)$ are as shown below

R_3

*COUNTRY	SEX	*AGE
{USA,CAN}	M	{11, ,40}
{USA,CAN}	M	{41, ,70}
{USA,CAN}	F	{11, ,40}
{USA,CAN}	F	{41, ,70}

R_4

*COUNTRY	*SEX	*AGE
{USA,CAN}	{M,F}	{11, ,40}
{USA,CAN}	{M,F}	{41, ,70}

Unpack (UN)

Let R ∈ D, A ∈ Atr(R), C_A = Atr(R) - {A} For each tuple t ∈ R, we define a set of tuples

$$UN_A(\{t\}) = \begin{cases} \{t\} \ \textit{if A is simple-valued} \\ \{t' \mid t'[A] \in t[A] \textbf{ and } t'[C_A] = t[C_A]\} \\ \textit{otherwise} \end{cases}$$

then $UN_A(R) = \bigcup_{t \in R} (UN_A(\{t\}))$

If A is simple-valued then $UN_A(R) = R$, otherwise $UN_A(R)$ maps each tuple t in R into a set of tuples such that each element in t[A] becomes the A-value of one resulting tuple

Example 10. Consider the relation R_1 in Fig 2 c, and relations R_3 and R_4 in example 9 Then

$UN_1(R_3) = \Pi_{1,2,3}(R_1)$, and $UN_2(R_4) = R_3$

Aggregate Formation [Klug82]

Let R ∈ D, X ⊆ Atr(R), | X | = k Let f be an aggregate function, and A, A ∈ Atr(R), be simple-valued Then $R<X, f_A>$ is a relation with degree k+1, and is defined as

$R<X, f_A> =$
{ t[X] o y | t ∈ R **and**
y = f_A ({t' | t' ∈ R **and** t'[X]=t[X]})}

where 'o' denotes concatenation

The aggregate formation operator first *partitions* tuples of relation R such that tuples having the same X component are in the same partition Then the function f is applied to component A of tuples in each partition, and the X-value and the associated aggregate value are output for each partition

Example 11. Consider the relation R_1 in fig 2 c The relation resulting from $R_1<\{1\}, SUM_4>$ is shown below

COUNTRY	SUM-COUNT
USA	330
CAN	190

Construct (CT)

This operation constructs a single-column set-valued relation using *incremental* (I) or *actual* (A) assignments Incremental assignment specifies ranges of values to form tuple components Actual assignment specifies tuples of the relation explicitly Construct operation is useful for creating on-the-spot category attribute instances

Example 12. The operation $CT_{(I\ [0,49,5])}$ creates a relation with tuples ({0,1,2,3,4}), ({5,6,7,8,9}), ,({45,46,47,48,49}) The operation $CT_{(A\ F,M)}$ creates a relation with tuples (F) and (M)

Aggregation-by-Template

Let R_1, R_2 ∈ D, Y ⊆ Atr(R_1), Z=Atr(R_2), where | Y | = | Z | ≥ 1 and each attribute in Z is set-valued Let Y_a be the set of attributes in Y that are simple-valued, and $Y_n = Y - Y_a$ Z_a and Z_n denote those attributes in Z that correspond to Y_a and Y_n Then $R_1<X, Y, f_A>R_2$ is a relation with degree | X | + | Y | + 1, and is defined as

$R_1<X, Y, f_A>R_2 =$
{ toy | ($\exists t_1$)($\exists t_2$)
($t_1 \in R_1$ **and** $t_2 \in R_2$ **and**

$t[X]=t_1[X]$ **and** t[Z] $= t_2[Z]$ **and**
y $= f_A$ ({ t′ | t′$\in R_1$ **and**
t′[X]=t[X] **and**
t′[Y_a]$\in$ t[Z_a] **and**
t′[Y_n] $\subseteq$ t[Z_n]}))}

The aggregation-by-template operator $R_1<X,Y,f_A>R_2$ groups tuples of R_1 as follows Let t be a tuple over attributes (X ∪ Z) such that t[X]=t_1[X] for some t_1 in R_1, and t[Z]=t_2[Z] for some tuple t_2 in R_2 Each such tuple t defines a group G_t of tuples of R_1 such that v, v $\in R_1$, is in G_t if v[X]=t[X], $v[Y_a]\in t[Z_a]$ and $v[Y_n]\subseteq t[Z_n]$ Then f is applied on attribute A of tuples in G_t The value returned by f applied over an empty group is null

Example 13. Consider the relation R_1 in fig 2 c The operation

$R_1<\{2\},\{1\},SUM_4>CT_{(A\ \{USA\},\{USA,CAN\})}$

produces the following relation where the new column is named SUM-COUNT

SEX	*COUNTRY	SUM-COUNT
F	{USA}	145
M	{USA}	185
F	{USA,CAN}	245
M	{USA,CAN}	275

The aggregation-by-template is more convenient than the aggregate formation when there are pre-specified groupings of attributes for aggregation (common in statistical databases) Also, the aggregation-by-template is based on grouping tuples (i e a tuple may belong to more than one group) while the aggregate formation is based on partitioning tuples However, each aggregation operator is expressible by an algebra expression utilizing the other aggregation operator [OzsO83]

5. Physical Organization of Summary Tables

If, for any subtree T in Fr or Fc, there is always a unique instance in a summary table, we say that the summary table has a *full cross product* Consider 1980-COUNT instance in fig 1 This summary table does not have a full cross product since the instance of GAME (Alpine-skiing) as a subtree of USA is different than the instance of GAME (Nordic-skiing) as a subtree of CANADA However, if we have GAME instances Alpine-skiing and Nordic-skiing as children of USA and CANADA (which amounts to adding two more rows to the summary table) then the resulting summary table does have a full cross product

When a summary table does not have a full cross product (i e an *incomplete cross product*) we may always extend the two dimensional array of cell values (i e, *cell value array*) with additional rows and columns such that cell attribute values in these new rows and columns are null denoting the absence of that cell from the instance of the summary table This way we expand an incomplete cross product into a full cross product The addition of null values increases the storage requirements, therefore compression techniques to remove the null values are employed [EggS 80, Mata 84]

In this section we consider the storage of summary tables with full cross product We separately store Fr, Fc, instances of each attribute in Fr and Fc, and the cell value array, and use computation to derive the associated cell attributes and their instances for a given cell attribute value in the cell value array

The rows of the array correspond to root-to-leaf path instances of trees in Fr, and the columns to root-to-leaf path instances of trees in Fc We assign to each row or column a specific root-to-leaf path instance Section 5 1 discusses the storage of category attributes and the assignment of root-to-leaf path instances to the rows and columns in the cell value array The storage of cell attribute names and cell values is discussed in section 5 2

5.1 Computing Category Attribute

Values of a Cell Position

For convenience, we transform Fr and Fc into ordered trees TR and TC by making the root nodes in Fr and Fc immediate descendants of dummy attributes θ_r and θ_c respectively θ_r and θ_c each has a single null value in its domain

A *tree instance* for TR is a tree where

(i) the nodes represent values for the attributes in TR,

(ii) values for attribute B are immediate successors of values for attribute A if and only if B is an immediate successor of A in TR

We call a root-to-leaf path of a tree instance a *root-to-leaf path instance* We say node A is *to the left* of node B in a tree if A is reached before B when the tree is traversed in preorder

If a tree instance is a full cross product or expanded into a full cross product by adding the missing root-to-leaf path instances then given a root-to-leaf path instance we can compute the corresponding row or column number in the cell value array and vice versa For the remainder of this section we assume tree instances with a full cross product, and give the formula to obtain category attribute values of a given cell attribute value We start with some definitions

Given an ordered subtree with root R the total number of leaves in the instance of the subtree with root R, denoted by C(R), is

$$C(R)=\begin{cases} |R| \text{ if R is a leaf} \\ |R| \sum_{i=1}^{n} C(S_i) \text{ otherwise} \end{cases}$$

where $|R|$ denotes the number of instances of attribute R and $S_1, S_2, \ldots, S_n$ are immediate successors of R

Let attribute S be an immediate successor of attribute R in an ordered tree The total number of leaves in the instance of a subtree with root R *and* to the left of S, denoted by L(R,S), is

$$L(R,S)=\begin{cases} 0 \text{ if S is the leftmost immediate successor of R} \\ \sum_{i=1}^{n} C(S_i) \text{ otherwise} \end{cases}$$

where $S_1, S_2, \ldots, S_n$ are immediate successors of R and they are to the left of S in the ordered tree

We denote the total number of leaves in the instance corresponding to one subtree of R, as S(R), where

$$S(R) = C(R) / |R|$$

Example 14. Consider the tree θ (COUNTRY (SEX,GAME) and its instance θ (USA (M, F, Alpine-skiing, Nordic-skiing), CAN (M, F, Alpine-skiing, Nordic-skiing)) Then

C(θ)=8, C(COUNTRY)=8, C(GAME)=2,
C(SEX)=2, L(θ,COUNTRY)=0
L(COUNTRY,GAME)=2, L(COUNTRY,SEX)=0
S(θ)=8, S(COUNTRY)=4, S(SEX)=1, S(GAME)=1

In order to represent attributes (set-valued and simple-valued) efficiently, we induce a total ordering to the values of a category attribute, and use the relative position i of an attribute value as its *encoded value*

Example 15. Consider COUNTRY attribute and its ordered instances USA, CAN We use integers 0 and 1 to encode USA and CAN respectively

Let (P,V) be a root-to-leaf path instance such that

(i) P=($X_1, X_2, \ldots, X_n$) is a root-to-leaf path of TR or TC, where the root is omitted, X_1 is an immediate successor of the root and X_{i+1} is an immediate successor of X_i for $1 \leq i < n$,

(ii) V=($x_1, x_2, \ldots, x_n$) is an ordered set of values, where x_i, $x_i \geq 0$, is an *encoded value* for the attribute X_i, $1 \leq i \leq n$

The leaf (i e , the row or column) number corresponding to the root-to-leaf path instance

(P,V), denoted as $D((X_1,x_1), \ldots ,(X_n,x_n))$ or simply D, can be computed as

$$D((X_1,x_1), \ldots ,(X_n,x_n)) = \sum_{i=1}^{n} (L(X_{i-1},X_i) + x_i S(X_i)),$$

where X_0 is the dummy root attribute of the ordered tree Note that leaves are numbered beginning at 0

Example 16. Consider the tree and its instance in example 14 The attribute values USA, M and Alpine-skiing have the encoded value 0, and the attribute values CAN, F and Nordic-skiing have the encoded value 1 Let P=(COUNTRY,SEX) and V=(1,1) (i e , V corresponds to the path (CAN,F)) Then

D((COUNTRY,1),(SEX,1)) = L(θ,COUNTRY) + S(COUNTRY)+L(COUNTRY,SEX)+S(SEX) = 5

Given a leaf number, D, the root-to-leaf path instance corresponding to D can be obtained as follows Let $\{H_1,H_2, \ldots ,H_n\}$ be the ordered set of immediate successors of X_0 (the dummy root attribute) Then

a) X_1 is equal to H_k where

$$L(\theta,H_k) \leq D < L(\theta,H_{k+1}) \quad \text{if } k \neq n,$$
and
$$L(\theta,H_k) \leq D \quad \text{if } k = n$$

b) The value for x_1 is given by

$$x_1 = \lfloor (D - L(\theta,X_1))/S(X_1) \rfloor$$

Now let $\{H_1,H_2, \ldots ,H_n\}$ be the ordered set of immediate successors of X_{j-1}, $j > 1$

a) X_j is equal to H_k where

i) $$L(X_{j-1},H_k) \leq D - \sum_{i=1}^{j-1} L(X_{i-1},X_i) - \sum_{i=1}^{j-1} x_i S(X_i) < L(X_{j-1},H_{k+1}) \quad \text{if } k \neq n,$$

ii) $$L(X_{j-1},H_k) \leq D - \sum_{i=1}^{j-1} L(X_{i-1},X_i) - \sum_{i=1}^{j-1} x_i S(X_i) \quad \text{if } k = n,$$

b) The value for x_j is given by

$$x_j = \lfloor (D - \sum_{i=1}^{j} L(X_{i-1},X_i) - \sum_{i=1}^{j-1} x_i S(X_i))/S(X_j) \rfloor$$

Example 17. Consider the tree and its instance given in example 14 Given D=2, the corresponding path attributes and their values are X_1=COUNTRY (since L(θ,COUNTRY)=0 $\leq$ D=2), x_1=0 (i e , USA) (since $\lfloor$ (2 - L(θ,COUNTRY))/S(COUNTRY)$\rfloor$ = 0), X_2 = GAME, and x_2 = 0 (i e , Alpine-skiing)

TR and TC are stored by using one (logical) record per node and links Each record contains the name of the attribute, the number of elements in the domain, a pointer to a table where the values of the attribute can be found, and pointers for the tree

5.2. Storage of Cell Attribute Names and Cell Values

We store the names of cell attributes in an array, called *the cell name array,* where the rows correspond to root-to-leaf paths of TR and the columns correspond to root-to-leaf paths of TC

The cell value array is stored in a *cell value file* in row-order with one value per (logical) record

Let <(PR,VR),(PC,VC)> be an ordered set of root-to-leaf path instances where

(i) PR=($X_1, \ldots , X_m$) is a root-to-leaf path of TR and θ_r is omitted from the path,

(ii) VR=($x_1, \ldots ,x_m$) is an ordered set of values for the attributes in the path PR, where x_i is an encoded value for X_i, $1 \leq i \leq m$,

(iii) PC=($Y_1, \ldots , Y_n$) is a root-to-leaf path of TC and θ_c is omitted from the path, and

(iv) VC=($y_1, \ldots ,y_n$) is an ordered set of values for the attributes in the path PC, where y_i is an encoded value for Y_i, $1 \leq i \leq n$

The relative number, r, of the record in the cell value file containing the cell value associated with <(PR,VR),(PC,VC)> is obtained as

$$r = D((X_1,x_1), \ldots ,(X_m,x_m)) \times C(\theta_c) + D((Y_1,y_1), \ldots ,(Y_n,y_n)) + 1$$

Given r, the relative number of a record in the cell value file, we obtain the ordered set of root-to-leaf path instances <(PR,VR),(PC,VC)> corresponding to r as follows

1) First we obtain the row number for (PR,VR), DR, and the column number for (PC,VC), DC

$$DR = \lfloor (r-1)/C(\theta_c) \rfloor \quad DC = r-1-DR\ C(\theta_c)$$

2) Then we obtain the root-to-leaf path instances corresponding to DR and DC (described in section 5 1)

6. Implementing Summary Table Operations

The aim of this section is to show briefly how the basic summary table operations are performed using the storage method presented in section 5 Below we assume that cell value compression is not used, and discuss only concatenate, extract and merge operations The other operations are similar The case when compression is used is discussed in [Mata84] We also briefly mention an improved storage method in section 6 1

Concatenation

To obtain the row concatenation of two summary table instances S_1 and S_2, in this order, we append the cell value file of S_2 to the cell value file of S_1 For the column concatenation of two cell value arrays S_1 and S_2 the n-th row of the cell value array for S_2 is appended to the n-th row of the cell value array for S_1 The complexity of row concatenate and column concatenate is $O(C(\theta_r) \times C(\theta_c))$

Extract

Let S be a summary table instance, T_1 be a subtree of θ_r and T_2 be a subtree of θ_c Assume X and Y respectively are the roots of T_1 and T_2 The process of extracting the summary table associated with the row subtree T_1 and the column subtree T_2 consists of locating in the original cell value array the row corresponding to the first root-to-leaf path instance of T_1, reading only those cell values corresponding to T_2 , and then writing them into the cell value file which contains the cell value array of the summary table extracted This process is repeated for each root-to-leaf path instance of T_1 The time complexity of this process is O(C(X) x C(Y))

Merge

To perform the column merge operation we read the rows of the original cell value array from the first row to the last row, one row at the time Once a complete row has been read we relocate the cell attributes in the row according to the column tree produced by the column merge operation and write the resulting row into a file containing the new cell value array Similarly, the row merge operation is performed by copying the rows of the old cell value array into a file containing the new cell value array, in the order established by the new row tree

6.1. An improvement

We have seen above that the execution of attribute merge and split operations relocate the rows or columns in the cell value array This occurs since the change in the schema also changes the order of the root-to-leaf path instances assigned to a row or column of the cell value array Nevertheless we can modify the storage method of section 5 in such a way that the cell value array does not need to be changed whenever a summary table scheme is changed by an attribute merge or split operation In the original method the summary table scheme is stored as two ordered trees, where the nodes of the trees contain names of attributes and the number of values in the domain of each attribute If we also include in the node for the attribute Y the values L(X,Y), where X is the immediate predecessor of Y, and S(Y), both obtained from the cell value array, then the cell value array does not need to be changed when

attribute merge or split operations occur The details of this approach are in [Mata84]

7. Conclusion

The summary table physical organization techniques discussed in this paper are being implemented within a prototype DBMS, called the System-for-Statistical-Databases (SSDB) [OzO284] The dictionary of SSDB is expanded to maintain (among other things) cell (attribute) name array, TR and TC of a summary table Cell value arrays of summary tables are kept in a direct-access file The query language of SSDB is a graphical, user-friendly language, called Summary-Table-by-Example (STBE) [OzsO84], which allows manipulation of set-valued relations and summary tables The storage structures of set-valued relations and summary tables, and the relational algebra operators for set-valued relations are implemented and operational Presently we are implementing a display manager and a query optimizer for STBE

References

[CCA79] Computer Corporation of America, File Manager's Technical Reference Manual, Model 204 DBMS System, Cambridge, Mass , 1979

[EggS80] Eggers, S J , Shoshani, A , "Efficient Access of Compressed Data", *Proc , VLDB Conf*, 1980

[IkeK81] Ikeda, H , Kobayashi, Y , "Additional Facilities of a Conventional DBMS to support Interactive Statistical Analysis", *Proc , First LBL Workshop on Statistical Database Management*, 1981

[Jaes82] Jaeschke, G and Schek, H J , "Remarks on the Algebra of Nonfirst Normal Form Relations", *Proc , ACM Symposium on Principles of Database Systems*, 1982

[Klug82] Klug, A , "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions", JACM, 29, 3, July 1982

[Mata84] Mata, F , "Physical Organization Techniques for Set-Valued Relations and Summary Tables", M S Thesis, CWRU, May 1984

[McCA82] Mc Carthy, J L , "Metadata Management for Large Statistical Databases", Proc , VLDB Conf , 1982

[Nie75] Nie, H N , et al, *SPSS-Statistical Package for the Social Sciences*, McGraw-Hill, 1975

[Olke82] Olken, F , "How Baroque Should a Statistical Database Management System be?", *Proc, Second LBL Workshop on Statistical Database Management*, 1983

[OzsO83] Ozsoyoglu, Z M , Ozsoyoglu, G , "An extension of Relational Algebra for Summary Tables", *Proc , Second LBL Workshop on Statistical Database Management*, 1983

[OzsO84] Ozsoyoglu, Z M , Ozsoyoglu, G , "Summary-Table-By-Example A Database Query Language for Manipulating Summary Data", *Proc , IEEE COMPDEC Conf*, 1984

[OzO284] Ozsoyoglu, G , Ozsoyoglu, Z M , "SSDB - An Architecture for Statistical Databases", *Proc , 4th International Joint Conference on Information Technology*, 1984

[Psta81] P-STAT Users Manual, Princeton, New Jersey, 1981

[Shos82] Shoshani, A "Statistical Databases characteristics, problems and some solutions", *Proc , VLDB Conf*, 1982

[SuNB82] Su, S , Navathe, S B , Batory, D S , "Logical and Physical Modeling of Statistical/Scientific Databases", *Second LBL Workshop on Statistical Database Management*, 1983

[Ullm82] Ullman, J D , *Principles of Database Systems*, Second Edition, Computer Science Press, Potomac, MA, 1980

[Uslb80] Bureau of Labor Statistics, "Table Producing Language System", Version 5, Washington, D C , 1980