A Data Model and Access Method for Summary Data Management

Meng Chang Chen and Lawrence McNamee

Computer Science Department
University of California, Los Angeles

Abstract
This paper proposes a data model and an access method for summary data management. Summary data, represented as a trinary tuple <statistical function, category, summary>, is meta-knowledge summarized by a statistical function of a category of individual information typically stored in a conventional database. For instance, <average-income, female, $45,000> is a summary data. The concept of category (type or class) and the additivity property of statistical functions form the basis of this model that allows for the derivation of new summary data. The complexity of deriving new summary data has been found computationally intractable in general and the proposed summary data model, with disjointness constraint, solves the problem without the loss of information. The proposed access method called Summary Data tree, or SD-tree, which handles an orthogonal category as a hyper-rectangle, realizes the proposed summary data model. The structure of the SD-tree provides for efficient operations including summary data search, derivation and insertion on the stored summary data.

1. Introduction
The advantages and analyses of using precomputed results to answer complex queries have been discussed in considerable detail in [14,7,20,8,18]. But, studies of summary data management and data modeling concepts have only been initiated recently. In this paper we carefully study the characteristics of summary data and, then, propose a summary data model and an access method to support the store and operations on summary data. Some research results of the summary data model can be found in [4].

Conventional database management systems (DBMS) are designed to store and handle queries about individual data. When a query is about a category of a large number of individuals, a DBMS needs to scan through a large portion of the database exhaustively, and thus inefficiently. A good approach for solving this problem is by means of a summary data management system (SDMS) which stores and manages properly selected precomputed results, called summary data. This approach can achieve a remarkable improvement in performance for many applications. For instance, in applications, like decision making and statistical query processing, summary data can be used directly to answer questions, while in some other applications, summary data can be used in choosing an effective way to solve problems, e.g. conventional and semantic query optimization. The summary data model and access method proposed herein is designed to assure prompt responses to statistical queries (which are defined roughly as queries about a summary of a category of individuals, like "What is the average age of male programmers who eat quiche?"). The proposed data model can also serve as a bridge between database management systems and many applications involving sophisticated statistical functions.

Following the classification of Smith et al [16], summary data in our model consists of the "non-inheritable attributes" which are often formed by applying 'summarizing' functions to instances of a category. A category (called type or class in other papers) is a set of tuples (or objects) satisfying some conditions. While the definition of a category is static, the tuples of an instance of this category might vary as the database is updated. A statistical function, which is a set function, inputs a set of tuples and generates, in a normal situation, a real number result called a summary. Summary data contains a category, a statistical function, and a summary which is a result of the statistical function of an instance of the category. If summary data is either precomputed and stored or derivable from other summary data, the use of summary data in answering statistical queries can reduce the search time tremendously since there is no need to scan through the original database.

The amount of summary data, which could be very large, is determined by the granularity of a category and the number of additive statistical functions required, and is not proportional to the number of tuples in the database. The proposed model of summary data provides an approach to determine the necessary summary data and a method to derive new summary data. To achieve good performance, a novel access method is required. Since an orthogonal category which is a product of sets can be viewed as an n-dimensional hyper-rectangular object in n-dimension space, the access method should be capable of handling n-dimensional objects. Operations required in deriving summary data are more complicated than the ordinary n-dimensional search that mitigates the existing access methods from consideration. Due to the unique characteristics and functions of the summary data model, preserving the inherent characteristics of summary data speeds the search and derivation of summary data, and prevents the loss of information. In this paper, a data structure called Logical Summary Data tree (LSD-tree) can achieve the above goals by implementing the category hierarchy.
However, in order to broaden the usage of the LSD-tree, the data structure should be accommodated by the paged secondary memory. The SD-tree is an implementation of an LSD-tree on a paged device. The handling of page overflow in the SD-tree is different from other n-dimensional access methods. In the initiation of the index tree of the SD-tree, the insertion and deletion.

A relation instance is a subset of the cross product of all the domains. Figure 2.1 is an example of a relation instance which is used throughout the paper. The category "male employees in CS department" is represented as:

\[ N \times (CS) \times \{\text{male}\} \times \{1...100\} \times \{\text{Manager,Engineer,Secretary}\} \times \{15...100\} \]

where \( \times \) denotes cross product. For simplicity, a category can also be represented in a shorthand which removes subsets that are exactly their domains. Thus the above category can be abbreviated as \((CS)\times\{\text{male}\}\). The category instance of \((CS)\times\{\text{male}\}\) of the given relation instance is the set of employees with EMP# 121, 124, and 177. 

<table>
<thead>
<tr>
<th>EMP#</th>
<th>DEPT</th>
<th>Sex</th>
<th>Age</th>
<th>Position</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>AD</td>
<td>male</td>
<td>42</td>
<td>Manager</td>
<td>75k</td>
</tr>
<tr>
<td>006</td>
<td>AD</td>
<td>male</td>
<td>40</td>
<td>Manager</td>
<td>60k</td>
</tr>
<tr>
<td>014</td>
<td>AD</td>
<td>female</td>
<td>55</td>
<td>Secretary</td>
<td>35k</td>
</tr>
<tr>
<td>030</td>
<td>EE</td>
<td>female</td>
<td>42</td>
<td>Manager</td>
<td>60k</td>
</tr>
<tr>
<td>034</td>
<td>EE</td>
<td>female</td>
<td>35</td>
<td>Engineer</td>
<td>52k</td>
</tr>
<tr>
<td>057</td>
<td>EE</td>
<td>male</td>
<td>28</td>
<td>Engineer</td>
<td>45k</td>
</tr>
<tr>
<td>089</td>
<td>EE</td>
<td>male</td>
<td>23</td>
<td>Secretary</td>
<td>25k</td>
</tr>
<tr>
<td>095</td>
<td>EE</td>
<td>female</td>
<td>29</td>
<td>Engineer</td>
<td>40k</td>
</tr>
<tr>
<td>121</td>
<td>CS</td>
<td>male</td>
<td>49</td>
<td>Manager</td>
<td>62k</td>
</tr>
<tr>
<td>124</td>
<td>CS</td>
<td>male</td>
<td>40</td>
<td>Engineer</td>
<td>55k</td>
</tr>
<tr>
<td>143</td>
<td>CS</td>
<td>female</td>
<td>27</td>
<td>Engineer</td>
<td>34k</td>
</tr>
<tr>
<td>177</td>
<td>CS</td>
<td>male</td>
<td>31</td>
<td>Secretary</td>
<td>31k</td>
</tr>
</tbody>
</table>

Figure 2.1 A Relation Instance of Employee Data

2.2 Statistical Functions and Summary Data

Hereafter Real denotes the set of real numbers, and RS denotes the set of all relation instances of a relation scheme, R.

Definition S: \( \text{RS} \rightarrow \text{Real} \times \{\lambda\} \) is called a statistical function which summarizes a set of tuples against some attribute(s)\( \lambda \). The output of a statistical function is called the summary of the input relation instance using the statistical function. A statistical query is defined as a query about a summary. A summary data is a 3-tuple consisting of the name of a statistical function, a category, and a summary. 

Example 2.2 The statistical function, "average income", generates the summary, 39.3, from the category instance of \((CS)\times\{\text{male}\}\). For another statistical function, "sum of products of age and income", on the same category instance, the summary is 6199. The summary data of the above examples are, respectively, 

\(<\text{average income}, (CS)\times\{\text{male}\}, 39.3\rangle\) and
<br>
\(<\text{sum of products of age and income}, (CS)\times\{\text{male}\}, 6199\rangle\). 

\(\lambda\) is defined as: \(\forall \epsilon \in \text{Real}\cup\{\lambda\}, \lambda + \epsilon = \epsilon\). Apparently, \(S(\emptyset) = \lambda\), where \(\emptyset\) denotes empty set.
The advantage of using summary data in answering statistical queries is the reduction of response time, as conventional database management systems need to scan through raw databases. However, the amount of all possible summary data is of the order of the product of the sizes of all possible categories and the number of statistical functions. In the worst case, the number of all possible categories can be up to \( \prod_a 2^{(\text{domain}(a)))} \). The huge amount of potential summary data not only requires a large storage capacity and a lengthy processing time, but also complicates the integrity maintenance problem when the database is updated. Thus, it is not practical to store all possible summary data. Instead, in our model only selected summary data are stored which is then used to derive new summary data by applying inference mechanisms.

### 2.3 Additive and Computed Statistical Functions

The additivity property allows additive statistical functions to generate the summary of a set of tuples directly from the summaries of a partition of the set.

#### Definition

A statistical function, \( S \), is called additive if \( \forall r_1, r_2 \in RS, \text{ and } r_1 \cap r_2 = \emptyset \), s.t. \( S(r_1 \cup r_2) = S(r_1) + S(r_2) \), where \( (\text{Real}(\cup),+) \) forms a commutative group.

Ordinarily the operation \( + \) is an arithmetic addition. The requirement of a group guarantees the existence of an inverse for elements with respect to \( + \), which permits direct computation for updating. In fact, this additivity property is a homomorphism from a boolean lattice formed by disjoint sets to the set of real numbers and \( + \) with addition. The following proposition shows that if a statistical function is additive, given a relation instance, and if the summaries (of this statistical function) of two disjoint category instances are known, then the summary of the union of two disjoint category instances can be calculated directly from the known summaries. Thus, when deriving a new summary of an additive statistical function, there is no need to know the details of the relation instance from which the summary comes, but one only needs to know the categories and their associated summaries.

**Proposition 2.1** Given an additive statistical function \( S \), and relation instance \( R \), for categories, \( C_i, C_j, C_k \), and requiring that \( C_i \cap C_j = \emptyset \) and \( C_k = C_i \cup C_j \), then \( S(R \cap C_k) = S(R \cap C_i) + S(R \cap C_j) \).

**Proof**

Since \( C_i \cap C_j = \emptyset \) and \( C_k = C_i \cup C_j \), then, \( \forall R, (R \cap C_k) = (R \cap C_i) \cup (R \cap C_j) \). Following the additivity property of \( S \), we obtain the result \( S(R \cap C_k) = S(R \cap C_i) + S(R \cap C_j) \).

Proposition 2.1 also states that \( S(R \cap C_k) = S(R \cap C_i) + (S(R \cap C_j))^2 \). The additive functions include many functions like sum, square, sum, etc., so that maximum and minimum are not additive since \( (\text{Real}(\cup),+) \), maximum and \( (\text{Real}(\cup),+) \), minimum are not commutative groups. Though some functions, like average, are not additive, they can still be computed directly employing the summaries of other additive statistical functions or computed results of other "nice" statistical functions. For example, the average function can be computed directly by dividing the result of the "sum" function by the result of the "cardinality" function. We call these "nice" functions \textit{computed} statistical functions.

**Definition** A statistical function \( S \) is called \textit{computed} from the results of other additive or computed statistical functions \( S_1, \ldots, S_n \), which are called the component functions of \( S \), if \( S \) can be represented as:

\[
S(r_i) = f(S_1(r_i), \ldots, S_n(r_i)),
\]

where \( f : (\text{Real}(\cup),+) \rightarrow (\text{Real}(\cup),+) \), and there is no cycle (i.e., quasi-order) in the definition of computed functions.

The concept of summary data can be applied to very complex applications. For example, Ghosh in [8] shows how summary data can achieve real time linear regression analysis.

**Example 2.3** The \textit{covariance} \( S_{xy} \) is a computed statistical function, since

\[
S_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})
\]

\[
= \frac{1}{n-1} \left( \sum_{i=1}^{n} x_i y_i - \frac{1}{n} \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i \right)
\]

\( S_{xy} \) can be computed directly from the results of additive functions, including \( n \) (cardinality), \( \sum x_i \) (sum of products), and \( \sum y_i \) (sum).

### 2.4 Derivability

In this section, the computing procedures of summary data and the computational complexity of the derivation process is discussed.

**Definition** Let \( A \) and \( B \) be categories. A proper difference, denoted by \( \Theta \), is defined as \( A \Theta B = A - B \) if \( B \subseteq A \). A disjoint union, denoted by \( \Theta \), is defined as \( A \Theta B = A \cup B \) if \( A \cap B = \emptyset \). A category is derived from a set of categories, if it can be expressed as a finite expression consisting of categories in the set as operands, \( \Theta \) and \( \Theta \) as operators, and matched pairs of parentheses.

A category is equal to an expression if it is equal to the execution result of the expression, which can be considered as the operational semantics of the expression.

**Proposition 2.2** For a relation instance \( R_1 \) and an additive statistical function \( S \), if a category \( C_k \) can be derived from a set of categories \( C \), the summary of the category instance, \( S(C_k \cap R_1) \), can be also obtained from the summaries or inverses of the summaries of the associated summary data.

**Proof** Given a set of categories \( C \), and a category \( C_k \), which is derivable from \( C \). Then \( C_k \) can be represented by a finite expression, \( EXP \), such that

\[
C_k = EXP(C_{i_1}, \ldots, C_{i_n}),
\]

where \( C_{i_j} \in C \) as operands, \( \Theta \) and \( \Theta \) as operators, and with matched parentheses.
Taking the intersection of \( R_1 \) with both sides of the equations, we obtain

\[ C_1 \cap R_1 = \text{EXP}(C_1 \cap R_1, \ldots, C_n \cap R_1). \]

This equation is still satisfied since \( \cap \) is distributive over \( \oplus \) and \( \otimes \), and any subexpressions in \( \text{EXP}(C_1 \cap R_1, \ldots, C_n \cap R_1) \) is still defined under \( \oplus \) and \( \otimes \).

Then, applying the additive statistical function on both sides, we arrive at

\[ S(C_1 \cap R_1) = S(\text{EXP}(C_1 \cap R_1, \ldots, C_n \cap R_1)). \]

Finally applying the following axioms iteratively, \( S(A \oplus B) = S(A) + S(B) \) and \( S(A \otimes B) = S(A) + S(B) - S(A \otimes B) \), to the expression, where \( A \) and \( B \) are subexpressions, the original expression eventually is replaced by an expression with \( S(C_j) \) or \( S(C_j)^{-1} \) as operands, addition as operations, and with matched parentheses. 

The above proposition indicates an approach for obtaining a new summary of additive statistical functions from precomputed summary data. However, proposition 2.3 below shows, in general, that this approach suffers from computational intractability.

**Proposition 2.3** To determine whether a category is derivable from a set of categories is an \( NP \)-hard problem.

**Proof** Can be found in [3].

### 2.5 Generating Category Set

The intractability problem results from the fact that categories overlap each other thereby creating decision points. Our solution to achieve a fast derivation is to decompose the overlapping category sets into disjoint category sets, which still preserve at least the same expressiveness, i.e., the decomposed disjoint category set can derive all the categories derivable from the former. In [4], it is proved that for two orthogonal categories of \( n \) attributes, there exists a set of at most \( 2^n + 1 \) disjoint orthogonal categories such that \( A \), \( B \), and \( A \cap B \) are derivable from the set.

**Definition** A disjoint category set \( C \) is called a generating category set of a relational scheme \( R \) if \( \forall t \in \text{Domain}(R), \exists C_i \in C \), such that \( t \in C_i \). A subset, \( MC \), of a generating category set is called the minimum cover of a category \( C_d \), if \( C_d \subseteq \bigcup C_i \) and \( \forall C_j \in MC, C_d \not\subseteq \bigcup C_j \).

**Example 2.4** This example shows how to calculate the covariance of Age and Income of the whole relation from summary data. Given a relation instance \( R_1 \) as in Figure 2.1 and a generating category set \( G \), \( \{g_1, g_2, \ldots, g_5\} \) where \( g_1 = \text{[AD]} \times \text{[male]}, g_2 = \text{[AD]} \times \text{[female]}, g_3 = \text{[EE]} \times \text{[male]}, g_4 = \text{[EE]} \times \text{[female]}, g_5 = \text{[CS]} \times \text{[male, female]} \). The summary data of categories, \( g_i \), and additive functions, Cardinality, \( \Sigma Age \), \( \Sigma Income \), and \( \Sigma Age \times Income \), are represented in tabular form as in Figure 2.2(a). The category equal to

\[ \text{Domain}(R) = \bigcup_{i \in S5} g_i. \]

For each additive function \( S \),

\[ S(\text{Domain}(R) \cap R_1) = \sum_{i \in S5} S(g_i \cap R_1). \]

The summary data of the whole relation scheme \( \text{Domain}(R) \), shown in Figure 2.2(b), can be obtained by summing up the summaries in Figure 2.2(a). Since the function of covariance is a computed function with component functions, Cardinality, \( \Sigma Age \), \( \Sigma Income \), and \( \Sigma Age \times Income \), the covariance can be obtained simply by substituting in the summary data of Figure 2.2(b). Figure 2.2(c) shows the calculation for covariance.

**Figure 2.2. Calculate the Covariance of Age and Income.**

In the proposed data model, each relation scheme has a corresponding generating category set. In addition, summary data for each category in the generating category set are precomputed and stored for some selected additive statistical functions. When a summary of a category is queried, an unique minimum cover is determined. Hence the queried summary can be obtained from the stored summary data, or a partial solution plus the unsolvable partial query is obtained.

**Example 2.5** Assume we have summary data as Figure 2.2(a). A query, "What's the total salary of female employees?", whose associated category has a minimum cover \( \{g_2, g_4, g_5\} \). Since the union of the minimum cover is not equal to its associated category, this query cannot be answered directly from the stored summary data. What can be obtained are a partial answer $187,000, which is obtained from the summary data of \( g_2 \) and \( g_4 \), and an unsolvable partial query "What's the total salary of female employees in the CS department?". This partial query can be solved by scanning the original database shown in Figure 2.1 with the answer $34,000. After inserting this new summary data, the summary data \( g_5 \) is replaced by
3. Summary Data Tree (SD-tree)

In a real application, the Data Base Administrator (DBA), during the database design phase, compiles the demands from users and generates a set of categories and statistical functions which are frequently utilized. An n-attribute orthogonal category can be represented as a hyper-rectangle in n-dimension space [12]. Thus, the proposed attribute orthogonal category can be represented as a hyper-statistical functions which are frequently utilized. An n-objects. There have been some research concerning n-dimension access methods in [1, 13, 11, 10, 9, 2, 15]. Since K-D-Tree [1], K-D-B-Tree [13], BD-Tree [11], and Cluster-Tree [2] are designed for handling point data, they are incapable of handling summary data efficiently. Grid file [6] divides the space uniformly which is not suitable for hyper-rectangles of an arbitrary size. R-Tree [9] allows overlaps in the intermediate nodes which will raise the complexity problem as discussed in Proposition 2.3. R*-Tree [15] is the one closest to our needs. However, R*-tree does not preserve the logical structure of the categories collected by the DBA since R*-tree's Algorithm Sweep packs pages of disks along the axes during the initiation of the index tree that might scatter a category or related categories among several pages on several different paths of a R*-tree. This undesirable phenomenon increases the time to derive summary data. In addition, while a non-leaf category is partitioned by the SplitNode operation of R*-tree after the index tree is generated, the associated summary of the partitioned category will be lost. This deleterious phenomenon causes the loss of information. The design of the SD-tree aims to overcome the drawbacks of R*-tree. There are also some access methods like SIAM [6] and TBSAM [17] that are also designed for handling summary data, but have been designed with a different philosophy in mind.

The proposed access method, SD-Tree, has an index tree of the category hierarchy and a data file of summaries. In order to realize the proposed summary data model, the functions that this access method has to support are:

1. **Summary data query handling.** Search and/or derive summary data when queried. If the queried summary data is not obtained, the system returns a partial solution and unsolved portion of the query (as shown in Example 2.5.)

2. **Consistency maintenance.** Update summary data when the database is updated.

3. **Summary data insertion and deletion.** Insert/delete summary data.

   The functions can be broken down into some lower level operations on the index tree which are:

   1. **Category search.** Find the minimum cover of the queried category.

2. **Insertion and deletion of a category.** Maintain the consistency of the index tree when a category is added/deleted.

   The SD-tree can handle both interval and non-interval attributes, although examples and algorithms given are for interval attributes only for simplicity. The graphs drawn in the examples are 2-dimensional for ease of reading; however, readers can easily extend the examples and algorithms to an n-dimension space. In section 3.1, 3.2, and 3.3, the generation of an SD-tree from a collection of categories is illustrated. The operations on an SD-tree are discussed in section 3.4.

3.1 Logical Category Block (LCB)

An LCB is a data structure representing a one-level category hierarchy. We identify an LCB by the name of the largest category, \(C_k\), which has to be orthogonal. Each item in an LCB contains, either a member category, \(C_{kj}\), or a residue, \(residue_k\), is a category, and has a pointer to the associated summary in the data file or to the associated LCB. Items that have pointers to summaries are called leaves. Each \(C_{kj}\) is an orthogonal category which is contained in \(C_k\) and disjoint from other member categories. The residue \(residue_k\) is equal to \(C_k - C_{k1} - \cdots - C_{kn}\) that is not necessarily orthogonal, and it is a leaf. As a special case, a category can be an LCB. For better performance, the whole LCB is expected to reside in the same physical page of a secondary device.

3.2 Constructing a Logical Summary Data tree (LSD-tree)

An LSD-tree is a tree having LCBs as its nodes. In an LSD-tree, a node is a member category of its father, and every pair of nodes is disjoint unless they have an ancestor-descendant relation. Figure 3.1 is an example of LSD-tree. Two categories in a collection of categories are called connected if there exists a sequence of categories in the collection such that they are the head and tail of the sequence respectively, and each category in the sequence overlaps its neighbors. A cluster is a collection of categories such that every two of the categories is connected. An orthogonal category is called the minimum region of a collection of categories if no smaller orthogonal category can cover the collection of categories. It is desirable to have a cluster of categories stored as close as possible for the reason that less disk I/O are needed in order to search/derive categories from the cluster.

Since the set of categories from the DBA is not likely to be mutually disjoint, decomposition of overlapping categories is demanded. Given a set of orthogonal categories, the algorithm Generate-LSD-Tree, shown below, can be employed to construct the associated LSD-tree.

Algorithm Generate-LSD-Tree

**Step 1.** [Find minimum regions.]

Divide the given set of categories into disjoint clusters, and find the minimum region for each cluster.

If two regions overlap, replace these two regions by their minimum region.
Step 2. [Generate an LSD-tree for each region.]
Generate an LCB for each category in the input collection.
For each region resulting from step 1, decompose each pair of overlapping LCBs in the region using the algorithm Decompose (shown below) recursively until there are no more overlapping categories.
Then, generate an LSD-tree by creating an LCB as the root and taking every LSD-tree in the region as subtree.

Step 3. [Generate the final LSD-tree.]
If there are too many LSD-trees generated in Step 2, group those LSD-trees and find the mutually disjoint minimum regions such that each region covers a group, then generate an LSD-tree for each region.
Apply the above operation recursively until the final LSD-tree covering the whole domain space is generated.

Figure 3.1 Example of LSD-tree

The algorithm Decompose decomposes a pair of overlapping LSD-trees into several disjoint LSD-trees.

Algorithm Decompose
Let A and B be the root LCBs of the given pair of overlapping LSD-trees.
Case 1. A ∩ B is not orthogonal. Without loss of generality, assume B is not orthogonal. Decompose B into a collection of orthogonal categories and make them sons of the father of B.
Case 2. A ∩ B is orthogonal. Pick one of A and B, say B. Decompose B-A into a collection of orthogonal categories, and along the decomposition, decompose the LSD-tree rooted by B into several LSD-tree, and make them subtrees of the father of B. Then, make A∩B as a subtree of A, and decompose A∩B and any other subtrees of A if they are not disjoint using this algorithm recursively.

Figure 3.2 Decomposition of Overlapping LCBs

3.3 Constructing a (Physical) Summary Data Tree (SD-Tree)

An SD-tree is a tree which results from confining an LSD-tree into a paginated data structure. Due to the limitation of a page size, some LCBs in an LSD-tree may have to be split into several physical pages. Nevertheless, the SD-tree still preserves the structure of the LSD-trees. Throughout the SD-tree generation, all pages with a packing ratio lower than minimum packing factor are put into the AVAIL pool for future use, and all the page requirements will check the AVAIL pool first for suitable pages before requesting secondary memory management. The following algorithm, Generate-SD-Tree, generates a SD-tree from an LSD-tree.

Algorithm Generate-SD-Tree
Step 1. [Pack subtrees of the LSD tree.]
Starting from the lowest subtrees of the LSD-tree, pack the whole tree into a block.
Stop packing before the resulting block that exceeds the pre-specified maximum packing factor. Store this block in a disk page.
Step 2. [Partition the over-size LCBs.]
If there is an LCB that exceeds the maximum packing factor, partition the LCB into several LCBs. The partition should avoid splitting the subtrees with longer paths.
If a subtree is split, propagate the split down to its nodes.
Step 3. [Pack the low packing factor pages.]
After all the LCBs in the LSD tree have been processed, merge the pages in the AVAIL pool if the resulting pages do not exceed the maximum packing factor, and update pointers.
3.4 Operations on SD-tree

The handling of a page overflow, i.e., an LCB is larger than a physical page, are different between during and after the initiation of the SD-tree. During the initiation, splitting overflow LCBs is used which will not result in loss of information since the summaries have not been collected yet. However, splitting overflow LCBs might lose information after all the summaries of the SD-tree have been collected. The introduction of a temporary LCB(TLCB) to solve the overflow problem is applied after the initiation of a SD-tree. The difference between a TLCB and an LCB is that the member categories of a TLCB are not necessarily disjoint and the summaries of its member categories are not necessarily complete. When an LCB overflows and has to be split, the items that are cut by the division are called cut items and others are called safe items. Since summaries of the cut items might be lost if the categories are split, the cut items should be freed from being split and stored intact. After the division, the summaries of the residues of safe member categories are still unknown. Figure 3.3 is an illustration of the creation of TLCB.

During the operation of the summary data management system, the functions including the search, insertion, and deletion of category are needed for operating the SD-tree. The category search operation can be used both for category search and derivation, since the operation always returns a minimum cover of the queried category. If a category is derivable from an SD-tree, the union of the returned minimum cover is exactly the queried category. The input of the algorithm Category-Search is an SD-tree rooted by R and a queried category C, and the output is a collection of disjoint categories, $MC = \{mc_i\}$, which is the minimum cover of C.

Algorithm Category-Search

Step 1. [Check termination.]
Collect R if R is a leaf, R overlaps C and R is not a residue with incomplete summary. Then return.

Step 2. [Traverse down the SD-tree.]
If an item $e_i$ of R has intersection with C, call Category-Search with input the subtree rooted by $e_i$ and C. 

The introduction of TLCB does not add any complication to category search. The TLCBs should be eliminated when the incomplete summaries of the partitioned items are known. The following is the algorithm for category insertion.

Algorithm Category-Insertion

Step 1. [Find the minimum cover.]
Use the algorithm Category-Search to find the minimum cover, $MC=\{mc_i\}$. 

Step 2. [Insert C.]
For each $mc_i$ overlapping C, divide the $mc_i$ into $mc_i \cap C$ and $mc_i - C$.

Case 1. $mc_i$ is a residue. Replace it by $mc_i - C$, and decompose $mc_i \cap C$ into a collection of orthogonal categories and make them as new items of the LCB containing $mc_i$.

Case 2. $mc_i$ is not a residue and $mc_i - C$ is not orthogonal. Generate a new LCB for $mc_i$, make $mc_i \cap C$ as the only item, and $mc_i - C$ as the residue.

Case 3. $mc_i$ is not a residue and $mc_i - C$ is orthogonal. Replace $mc_i$ by two new items $mc_i \cap C$ and $mc_i - C$. 

Step 3. [Handle Page Overflow.]
(Do this step if any page overflows after the insertion of C.)
If the overflowed page has more than one LCB, relocate some LCBs to pages in the AVAIL pool or to new pages. If the overflowed page only has one LCB, divide the LCB subject to the minimum number of cut item. Create a temporary LCB(TLCB) as shown in Figure 3.3 to store this anomalous situation.

The algorithm Category-Deletion deletes the category C from the SD-tree rooted by R, such that the summary of C is not revealed. Generally there is no need to delete a category.
except for data security.

Algorithm Category-Deletion
Step 1. [Find the minimum cover.]
Use the algorithm Category-Search to find the minimum cover, MC={mc}$^\prime$s are collected from the Category-Search.
Step 2. [Delete C, if found.]
If the union of MC is equal to C, delete all the items and the LCBs and the TLCBs formed solely by mc$^\prime$s.
Otherwise, the deletion fails. ☐

5. Conclusion
While a relational model provides a harmoniously integrated approach to data organization and list-type queries, statistical queries still have to be handled in an ad-hoc fashion. The proposed data model and access method allows summary data and statistical queries to be integrated within the relational model through the use of categories. The new class of queries can also be serviced in a reasonably performing fashion.

We are currently extending this work to include additional functional capabilities, and are planning to implement the proposed system through an existing relational DBMS. This will allow us to evaluate the utility and performance of the proposed system. The theoretical analysis of the performance of the SD-tree is part of the ongoing research in summary data management.

6. Acknowledgment
This research was funded by the IBM Almaden Research Center. The authors would also like to thank Dr. Kwan Wong and Dr. Sakti Ghosh of IBM, for their support and valuable discussions.

References