Exploring and Comparing Table Fragments With Fragment Summaries

Fatma-Zohra Hannou
Sorbonne Université, CNRS, LIP6
4, place Jussieu
75252 Paris, France
Email: Fatma.Hannou@lip6.fr

Bernd Amann
Sorbonne Université, CNRS, LIP6
4, place Jussieu
75252 Paris, France
Email: Bernd.Amann@lip6.fr

Mohamed-Amine Baazizi
Sorbonne Université, CNRS, LIP6
4, place Jussieu
75252 Paris, France
Email: Mohamed-Amine.Baazizi@lip6.fr

***Abstract*—In this article, we introduce a new pattern-based summarization framework for representing and *reasoning* about fragmented data sets. A *fragment summary* is a concise, complete and precise representation of a data fragment and its information contents *relatively to the whole data set*. We formally define the notion of fragment summary and the use of Structured Query Language (SQL) queries over fragment summaries for analyzing data fragments. We introduce an algorithm for computing summaries and present an experimental evaluation using two real-life data sets.**

***Keywords*–data fragments; summarization; patterns; reasoning.**

I. INTRODUCTION

Summarization is the process of generating a concise representation of a data set for some specific processing tasks. Compared to data compression, the goal of summarization is, usually, to preserve enough information for fulfilling these tasks without the need for decompression. Data summarization has been applied to various data types (text, data streams, graphs, structured data, music, etc.) for different tasks related to information retrieval, data monitoring, data visualization, query processing, and data integration as witnessed by recent works [1]–[5].

In this article, we introduce a new pattern-based summarization framework for representing and *analyzing* fragmented data sets. A *fragment summary* is a concise representation of a data fragment and its information contents *relative to the whole data set*. Data fragments can be the result of user-defined filtering queries or any other data extraction task over some data set. They can also correspond to data tables (sources) that partially cover a complete reference table like a dimension table in some analytic data set. Fragment summaries can then be used to rapidly *decide if a data tuple or a category of tuples (defined by attribute/value pairs) is completely or partially included* in the corresponding data fragment.

To better understand the potential of fragment summaries, consider **Energy** in Table I reporting the daily energy consumption for rooms (ro) in two floors (fl). The table contains a tuple for each location and time defined by the week (we) and the day (da), and indicates missing information with Null. The first data fragment is defined by query Q_{avail} returning all tuples with existing kWh values. The second fragment, denoted with Q_{miss}, contains all tuple identifiers with missing kWh values ; it is complementary to the first fragment. Both fragment summaries are presented in Table II. A fragment summary corresponds to a table of *patterns* that concisely summarizes all *data categories* contained in the fragment. The summary P_{avail} describes all complete categories of available information (for example, all values are available for category [we='w1'] and any sub-category), whereas the summary P_{miss} describes all empty categories (*e.g.* all values are missing for category [ro='r2', we='w2']). Observe that both fragment summaries contain the *minimal set of all attribute/value pairs* that are necessary to precisely characterize this fragment (and all categories it subsumes). For example, categories [fl='f1', ro='r1'] and [we='w2'] are not subsumed by any pattern in both tables and therefore contain tuples with missing and with available measures. Observe also that the intersection of both summaries is empty and the union is the "wildcard" template $*$ covering the whole data set.

TABLE I. DATA TABLE

Energy	fl	ro	we	da	kWh
t_0	f1	r1	w1	Mon	10
t_1	f1	r1	w1	Tue	12
t_2	f1	r1	w2	Mon	10
m_0	f1	r1	w2	Tue	Null
t_3	f1	r2	w1	Mon	8
t_4	f1	r2	w1	Tue	10
m_1	f1	r2	w2	Mon	Null
m_2	f1	r2	w2	Tue	Null
t_5	f2	r1	w1	Mon	12
t_6	f2	r1	w1	Tue	7
t_7	f2	r1	w2	Mon	8
t_8	f2	r1	w2	Tue	8

TABLE II. FRAGMENT SUMMARIES OF Q_{AVAIL} AND Q_{MISS}

P_{avail}	fl	ro	we	da
p_0	$*$	$*$	w1	$*$
p_1	f2	$*$	$*$	$*$
p_2	f1	r1	$*$	Mon

P_{miss}	fl	ro	we	da
p_3	$*$	r2	w2	$*$
p_4	f1	r1	w2	Tue

Fragments summaries are meant to provide a precise semantic characterization of data fragments and, thus, can be exploited for performing complex analytical tasks such as analyzing the *completeness* and *correctness* of analytic aggregation queries. To illustrate this case, consider the following query over the original data table **Energy**(fl, ro, we, da, kWh):

```
select fl, ro, we, sum(kWh) from Energy
 group by fl, ro, we
```

By examining this data table, it is easy to notice that the computed sum for the partition [f1,r1,w1] is be *incorrect* since this partition is *incomplete*. This examination allows inferring that no value is returned for the partition [f1,r2,w2] whose values are *missing*. Fragment summaries are very useful in supporting such analysis tasks especially when data becomes large and not amenable to visual inspection ; in this case, SQL turns out to be helpful as illustrated with the following queries over fragment summaries:

- **select** fl,ro, we **from** P_{avail} **where** da='*' retrieves the summary of all partitions with a correct aggregation result (e.g., the fragment characterized with week=w1 and floor=f2)
- **select** fl,ro, we **from** P_{avail} **where** da<>'*' returns the partitions with incorrect results (all results are incorrect for room 'r1' at floor 'f1') and
- **select** fl,ro, we **from** P_{miss} **where** da='*' characterizes the missing results (all results are missing for room 'r2' at week 'w2').

Fragment summaries characterize *all* complete categories in a fragment and their compactness ratio (summary size/fragment size) can be very high (see Section VI). This space loss is compensated by the speedup of complex decision tasks, such as identifying complete and missing data and annotating incorrect query results.

In our previous example, all fragments (and their summaries) were defined over a set of attributes that are part of the table *key*. In this case, the summary of any fragment is "complete" in the sense that it allows to decide for each tuple if it is part of the fragment or not. It is also possible to build summaries over sets of attributes that are insufficient to separate fragment tuples from non-fragment tuples. In this case, some tuple categories can be formally identified as belonging to some fragment, but other categories may be non-distinguishable.

To illustrate the notion of non-distinguishable (ND) fragments, consider the Adult dataset [6] that is widely used for learning population classes based on income. This dataset reports census data about income for $32,561$ individuals described by 14 attributes including one numerical attribute *Age* with domain $[17, 90]$ and seven categorical attributes *Workclass*, *Education*, *Marital-Status*, *Occupation*, *Race*, *Sex* and *Income* [<50k,≥50k]. In [7], this data-set was used for producing approximate summaries of highly frequent data categories. An example of such a summary is the one stating that 69% of individuals with *[race='White', sex='Male', Marital-Status='married-civ-spouse']* have a high-income (i.e., >50K). Instead of producing approximate summaries, we are interested in obtaining exact ones by exploiting additional attributes like: Age, $Workclass$, $Education$ and $Occupation$ and classify the individuals into High, Low and Non-Distinguishable (ND) as reported in Table III.

Here, we can see that (1) all white married male soldiers between 40 and 50 and all employees of the federal government with a PhD have a high income, (2) all white married males that have a low income, are young or have never worked or have a preschool diploma and (3) it is not possible to decide for white married males with a Master degree, or are old with a PhD, whether they belong to high or low-income class. We will show in Section V how this kind of detailed analysis can be done by evaluating simple SQL queries over pre-computed fragment summaries.

TABLE III. "MARRIED-WHITE-MALE" FRAGMENT

High

age	workc	education	occupation
40-50	$*$	$*$	Armed-Forces
$*$	federal-gov	Doctorate	$*$

Low

age	workc	education	occupation
< 20	$*$	$*$	$*$
$*$	Never-worked	$*$	$*$
$*$	$*$	Preschool	$*$

ND

age	workc	education	occupation
$*$	$*$	Masters	$*$
> 60	$*$	Doctorate	$*$

Contributions: In this article, we formalize the notion of fragment summary and show how fragment summaries can be exploited for characterizing data fragments in data sets and query results. Our approach leverages the relational representation of fragment summaries by taking advantage of SQL for achieving these different analyses. We introduce an algorithm for efficiently generating fragment summaries and sketch a mechanism for reasoning on fragment summaries to gain knowledge about data.

Outline: This article is organized as follows. In Section II, we survey related work before introducing our data model in Section III. Sections IV and V are dedicated to presenting the fragment generation algorithm and the reasoning mechanism, respectively. Experimentation results are discussed in Section VI, and we conclude the article in Section VII.

II. RELATED WORK

Our contribution lies in the intersection of two mainstream topics: data summarization and relative data completeness, which is a special case of data completeness. We report on works addressing both topics.

A. Data summarization

The main goal of most approaches in this family is to reduce the data size while preserving as much information as deemed useful for achieving specific operations like evaluating aggregate queries [4] or returning approximate answers with correctness guarantees [1]. Different techniques are used. Some approaches exploit semantic knowledge like fuzzy thesauri and linguistic variables [8][9] or OLAP hierarchies [2][3] to generate concise descriptions of large data sets. Efficient encoding of data has also been used for compressing columns and rows [10] but our work is more reminiscent to the family of *pattern mining* approaches [7][11][12] where summaries are expressed using patterns. In [7], summaries are built by selecting the most representative patterns that capture the largest fragments of data. The approach uses the Minimum Description Length principle for guiding the extraction process that searches for a minimal patterns-set with maximal informativeness. Differently from [7], our approach is concerned with extracting an *exhaustive* set of patterns characterizing a table fragment w.r.t. entire table and a set of attributes.

B. Relative data completeness

Information completeness is a major data quality issue that received attention in the database context [13]–[16]. Several approaches have addressed the problem of assessing the query answer completeness when queries are evaluated on a database with possibly missing tuples or Null values. Relative information completeness assumes the existence of a virtual or materialized reference database DB_C describing the full extent of data. The data set can be described by views over a virtual DB_C [15] and assessing the completeness of a query resorts to determining whether it can be answered using these views. In [17], data completeness is defined by a set of containment constraints between the database D and a master dataset DB_C, and D is complete for a query Q relative to DB_C, if adding tuples to D either violates some constraints or does not change the answer of Q.

The use of metadata for describing the data completeness has been investigated in [13][14][16][18]. *C-tables* [13] and *m-tables* [18] have been proposed for annotating tuples with certainty information and propagating certainty to query answers. In [16], patterns were used to annotate tables with completeness information and a pattern algebra was designed for reasoning on query answer completeness. Differently from [16], which assumes the existence of a set of patterns describing complete data fragments, our approach investigates how to efficiently extract such patterns from reference data and, more importantly, how to ensure that the extracted patterns exhaustively capture completeness information. Moreover, we do not restrict ourselves to the study of completeness but use patterns as means to characterize any data fragment with respect to a reference dataset and a set of attributes.

III. Data Model

In this section, we introduce the notion of fragment summary as a comprehensive description of all complete data categories in a data fragment. Let S and T be two relational tables such that $S \subseteq T$. Then S is called a *fragment* of *source or reference table* T and the pair $F = (S, T)$ is called a *constrained fragment*. For example, any table T with Null values for a given attribute A can be decomposed into two constrained fragments $F_{notnull} = (S_{notnull}, T)$ and $F_{null} = (S_{null}, T)$ where fragment S_{null} contains all tuples in T *with null values for* A and fragment $S_{notnull}$ contains all tuples in T *without null values*. In the following, we assume that the source table T of each constrained fragment is known and use without distinction the terms *fragment* and *constrained fragment*.

A. Fragment Summaries and Patterns

Let $A = \{a_1, a_2, ..., a_n\}$ be a set of attributes where the domain of each attribute is extended by a distinguished *wildcard* value $*$. A *pattern* $p = [a_1 : v_1, a_2 : v_2, ..., a_n : v_n]$ over A is a tuple that assigns to each attribute $a_i \in A$ a value $v_i \in dom(a_i) \cup \{*\}$ in the extended domain of a_i. A set of patterns $P(A) = \{p_1, p_2, \ldots, p_k\}$ over a set of attributes A is called a *pattern table*. We denote by $[*]$ the *wildcard pattern* where all pattern attributes are assigned to wildcards. Observe that a pattern table might contain only data tuples, i.e. patterns without any wildcards.

A pattern table P defines a hierarchy of patterns $L_F = (P^*, \leq)$ where $p \leq p'$ if p can be obtained from p' *by replacing zero or more constants by wildcards* (p is called a specialization of p') and P^* contains all patterns p' such that there exists a pattern $p \in P$ where $p' \leq p$ or $p \leq p'$. Then, the *pattern instance* $\lhd(p, S)$ of a pattern p over pattern attributes A in some table S is the sub-fragment or *category* of tuples $t \in S$ where $t[A] \leq p$ ($t[A]$ denotes the projection of t on attributes A). It is also easy to show that the following properties hold for pattern instances:

- $\lhd([*], S) = S$;
- $\lhd(p, \lhd(p, S)) = \lhd(p, S)$;
- $S \subseteq S' \Rightarrow \lhd(p, S) \subseteq \lhd(p, S')$.

The notion of instance can naturally be extended from patterns to pattern tables P and constrained fragments $F = (S, T)$: $\lhd(P, S) = \bigcup_{p \in P} \lhd(p, S)$ and $\lhd(P, F) = (\lhd(P, S), \lhd(p, T))$.

The following definitions introduce several properties for pattern sets that are necessary for defining the notion of fragment summary. Constrained fragments are related to pattern tables through the notion of *pattern satisfaction*. A constrained fragment $F = (S, T)$ *satisfies* a pattern p if the instance of p in the data table T is equal to the instance of p in the fragment S: $\lhd(p, T) = \lhd(p, S)$. We also say that pattern p *characterizes* fragment F. By extension, a constrained fragment F *satisfies* a completeness pattern table P if F satisfies all patterns in P. We can show that all fragments F_i in Section I satisfy the corresponding pattern tables P_i. A pattern table P *covers* a constrained fragment F if *for all patterns* p characterizing fragment F, there exists a pattern $p' \in P$ where $p \leq p'$. We can show that all pattern tables P_i in the introduction cover the corresponding fragments F_i. Observe that a pattern table P covering a constrained fragment F is not necessarily satisfied by F. In particular, all pattern table containing the universal pattern cover (but are not satisfied by) *all* constrained fragments. Then, a pattern table P *strictly covers* a constrained fragment F if P covers F and F satisfies P. Finally, a pattern table P is *reduced* if there exists no pair of distinct patterns $p \in P$ and $p' \in P$ such that $p' \leq p$.

Proposition 1: For each constrained fragment F, there exists a unique *reduced strict cover*, called the *fragment summary* of F, and denoted by $\rhd(F)$.

All pattern tables P_i in the introduction are fragment summaries of the corresponding fragments F_i.

First, observe that a fragment summary $\rhd(F)$ is not necessarily minimal with respect to instantiation, i.e., there might exist a subset of patterns $P' \subset \rhd(F)$ where $\lhd(P', F) = \lhd(\rhd(F), F)$ (all categories described $\rhd(F)$ are subsumed by the patterns in P'). This makes our summarization model different from other models, which try to maximize the compression ratio, whereas fragment summaries are compact representations of *all* characteristic data categories. Second, a fragment summary might only cover a strict subset of its fragment $F = (S, T)$, *i.e.*, $\lhd(\rhd(F), T) \subset F[A]$. We call the set of all tuples in $t \in F - \lhd(\rhd(F), T)$ the *rest* of F: $\mathcal{R}(F) = S[A] - \lhd(\rhd(F), F)$. The rest $\mathcal{R}(F)$ defines all categories of tuples in F that cannot be distinguished from other data tuples $t' \notin F$ by pattern attributes A, *i.e.* $t[A] = t'[A]$. This rest can again be considered as a new fragment that can be summarized. We also can easily show that, if A is a key in the source data table T, the rest is empty, *i.e.*, $\lhd(\rhd(F), F) = S[A]$ for all fragments $F = (S, T)$.

IV. Computing Fragment Summaries

Algorithm *FoldData* (Algorithm 1) computes for a given constrained table $F = (S, T)$ a strict cover $\rhd(F)$ over a set of attributes A. If A is the set of all attributes in F, *FoldData* produces the summary of F. The algorithm explores the data table by searching for complete sub-fragments (categories) that correspond to some specific pattern. It starts from the most general pattern *i.e.* wildcard pattern $[*]$ (level 0) and explores *top-down* and *breadth-first* the pattern subsumption lattice L_S generated by the active attribute domains in the data table S. Each level l corresponds to all patterns p with l constants. For checking if some pattern p is satisfied by S, the algorithm compares the size of the instances in p in S and T. After each level, all specializations of the derived complete patterns p are by definition also complete and the tuples covered by p can be pruned from S before executing the next level. Algorithm $FoldData$ uses the following functions:

- $powerSet(A, level)$ produces all sets of $level$ attributes in A.
- $patterns(A, S)$ produces for a set of attributes A all patterns $\pi_A(S) \times \{[*]\}$
- $checkComp(p, S, T)$ checks if $\lhd(p, S) = \lhd(p, T)$
- $prune(P, S)$ deletes from S all tuples satisfied by patterns $p \in P$.

Observe that operations $checkComp$ and $patterns$ can be implemented by standard SQL queries. In particular, $patterns$ is a simple projection on S and $checkComp$ can be implemented by comparing the result of two *count*-queries on S and T (we suppose that $S \subseteq T$). In the worst case, $FoldData$ explores

Algorithm 1: Algorithm $FoldData$

```
Data: constrained table $F = (S, T)$, attribute set $A$
Result: summary $\rhd(F)$
1 $P := \emptyset$ ; for $level := 0$ to $|A|$ do
2     $\mathcal{X} := \emptyset$ ;
3     for $B \in powerSet(A, level)$ do
4         for $p \in patterns(B, S)$ do
5             if $checkComp(p, S, T)$ then
6                 $P := P \cup \{p\}$ ; $\mathcal{X} := \mathcal{X} \cup \{p\}$ ;
7     $prune(\mathcal{X}, S)$ ;
8 return $P$
```

(almost) the whole pattern lattice L_S that is generated by all attribute×value combinations in the fragment. The number of patterns $size(L_S)$ of L_S depends on the active attribute domains in the fragment S and the number of attributes $n = |A|$: $size(L_S) = \sum_{i=1}^{n}(C_i^n) * D_i$ where D_i is the maximum size of the Cartesian product of the active domain of i attributes in the data table. The size of the source table influences the cost of checking pattern satisfaction. We also can estimate an upper bound for the fragment summary size as follows. Each tuple in the fragment generates between 0 (for tuples that are subsumed by patterns generated by other tuples) and k patterns, where k is the number of identifiers of the tuple in the source (reference) table. In the worst case, the size of the generated summary is $max_{1 \leq i \leq n} C_i^n \simeq C_{n/2}^n$ times the size of the fragment where $n = |A|$ is the number of attributes in A. Such a worst-case scenario corresponds to the particular case of random missing data with highly correlated attribute values and no pruning opportunities. If all attributes are necessary to identify any tuple in the source table (independent attribute domains), the fragment summary cannot get bigger than the fragment. As we show in our experiments, real-world data generally follows more regular incompleteness schemes, which increase the compression rate and folding performance.

V. Reasoning with fragment summaries

A. Formal reasoning model

Fragment summaries and fragment patterns in general are concise characterizations of data fragments and can be used for analyzing and comparing data fragments extracted from a given reference data set. By the previous definition of fragment summary, the following constraints hold for all constrained fragments $F = (S, T)$ where $T \neq \emptyset$ and all patterns $p \in \rhd(F)$ of their summaries :

- the instance $\lhd(p, S)$ is *complete* with respect to T and *not empty*.
- the instances $\lhd(p', S)$ of all specializations p' of p are complete with respect to T (but might be empty) and
- the instances $\lhd(p', S)$ of all generalizations $p' \neq p$ of p are *incomplete* with respect to T and *not empty*.

Similarly, let $\overline{F} = (\overline{S}, T)$ denote the *complement* of fragment F where $\overline{S}$ contains all tuples "missing" in S w.r.t. T and $\rhd(\overline{F})$ be the summary of $\overline{F}$. Then as before, we can show for all $F = (S, T)$ where $T \neq \emptyset$ and all patterns $p \in \rhd(\overline{F})$ in the summary of $\overline{F}$:

- the instance $\lhd(p, S)$ is *incomplete* with respect to T and *empty*.
- the instances $\lhd(p', S)$ of all specializations p' of p are empty (but might be complete) and
- the instances $\lhd(p', S)$ of all generalization $p' \neq p$ of p are incomplete with respect to T and not empty.

We can show that the rest of a fragment summary is equal to the rest of its complement $\mathcal{R}(\rhd(F)) = \mathcal{R}(\rhd(\overline{F}))$. We denote the summary of this rest of "indistinguishable" patterns by *ND*$(F, \overline{F}) = \rhd(\mathcal{R}(\rhd(F)) = \rhd(\mathcal{R}(\rhd(\overline{F}))$.

For all patterns p where there exists no generalization in $\rhd(F) \cup \rhd(\overline{F})$ the following holds :

- the intersection between the instance $\lhd(p, F)$ and both, F and $\overline{F}$ is not empty.
- if p is a tuple (only constant attribute values), then p and all generalizations of p are in *ND*$(F, \overline{F})$.

Based on these properties, all patterns in the summary of a fragment F can be classified into (1) C patterns, which have a complete instance in the fragment, (2) ND patterns, which have a incomplete and *indistinguishable* instance in the fragment, (3) E patterns, which have an empty instance in the source (reference) table and (4) IN patterns, which cover all other patterns. Recall from our introduction example in Section I the "White Male Married" fragment from the Adult dataset. Table III shows some patterns of each fragment summary of fragments "high income", "low income" and "indistinguishable" (ND). For simplification, we assume in the following that these tables represent the entire fragment summaries. Consider the patterns in Table IV. We want to reason and decide for each pattern the fragment belongs to.

Figure 1 is a tree representation of patterns where each node at some level i corresponds to a pattern of length i and each node corresponds to an attribute value at a given level. The wildcard pattern $[*]$ is the root; the first level

TABLE IV. ADULT DATASET PATTERNS

age	workc	education	occupation
$*$	$*$	$*$	$*$
< 20	$*$	$*$	$*$
$40 - 50$	$*$	$*$	$*$
$*$	$*$	masters	$*$
$*$	$*$	doctorate	$*$
$40 - 50$	$*$	preschool	$*$
> 60	$*$	doctorate	$*$
$*$	federal-gov	doctorate	$*$
$40 - 50$	$*$	$*$	Armed-force
$40 - 50$	never-worked	$*$	Armed-force

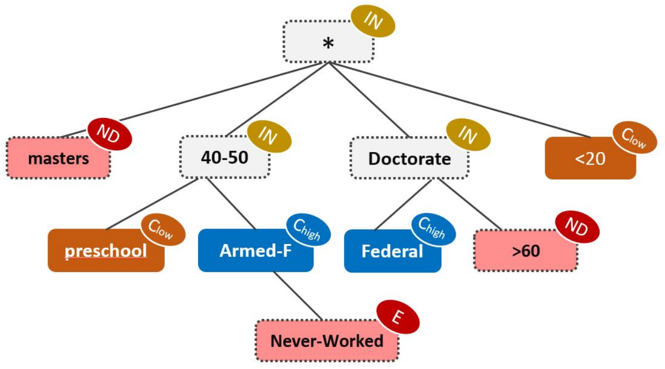


Figure 1. Labeled completeness pattern hierarchy

corresponds to patterns $[*, *, masters, *]$, $[40 - 50, *, *, *]$, $[*, *, doctorate, *]$ and $[< 20, *, *, *]$ with one attribute. The second level specializes the patterns at level one by replacing one wildcard by a constant. All patterns in summary $\triangleright(High)$ are complete and labeled by C_{high} (in blue) and all patterns in summary $\triangleright(Low)$ are labeled C_{low} (in orange). All ancestors of these patterns nodes are IN-patterns (incomplete, not empty, distinguishable). Patterns $[*, *, masters, *]$ and $[> 60, *, Doctorate, *]$ are indistinguishable (ND) (in red). Finally, pattern $[40-50, never-worked, *, Armed-Forces]$ is empty (label E) since it specializes a complete high income pattern $[40 - 50, *, *, Armed - Forces]$ and a complete low income pattern $[*, never - worked, *, *]$ (not shown in Figure 1).

B. Reasoning with SQL Queries

Let $RA_{ext} = RA \cup \{\triangleright, \triangleleft\}$ be the relational algebra extended by two operators $\triangleright$ and $\triangleleft$ where (1) $\triangleleft_A(P)$ generates for a given pattern table P an *equivalent* pattern table P' where all values of attributes $a_i \in A$ are constant values and (2) $\triangleright_A(P)$ generates for a given pattern table P an *equivalent* pattern table where there exists no pattern p and subset $S \subseteq P'$ with more than one pattern that is equivalent to p : $\nexists p, S \subseteq P', |S| > 1 : \{p\} \equiv S$. Using this extended algebra, we can define queries over fragment summaries. First we can define two operators $\triangleright(F) = \triangleright_A(F)$ and $\triangleleft(P) = \triangleleft_A(P)$ that compute the summary of some fragment F and the instance of a pattern table P, respectively. Unfolding $\triangleleft$ can directly be translated into the relational algebra by joining the pattern table with the data table, whereas folding $\triangleright$ over a set of attributes needs recursion, which is not expressible in relational algebra (see Section IV for implementations of $\triangleright$). Based on this formalization, it is then possible to rewrite any pattern query *without folding* into a relational SQL query over source tables and their fragment summaries. We will illustrate this by two examples.

First, selection can be applied for checking if some given pattern p is a specialization/generalization of a pattern $p' \in P$. For example, when considering the summary P in Table IV, pattern $[40 - 50, *, Doctorate, Armed - Forces]$ is complete (C) in fragment $High$ or empty(E) in the source table if the result of following query over the summary $\mathcal{P}(High)$ is not empty:

```
select * from P(High)
where (age='40-50' or age='*')
and (education='Doctorate' or education ='*')
and (occupation='Armed-Forces' or occupation='*')
```

It is easy to see that the result contains pattern $[40 - 50, *, *, Armed - Forces]$.

Joining two summaries needs unfolding. Consider two summaries $P_1(age, workc)$ and $P_2(workc, education)$ of two fragments S_1 and S_2 of data table Adult. The natural join of these two summaries generates a new summary $P(age, workc, education)$ characterizing the fragment $S_1 \bowtie S_2$:

```
select P1.age,T.workc,P2.education
from P1, P2, Adult
where (P1.age=Adult.age or P1.age=*)
and (P1.workc=Adult.workc or P1.workc=*)
and (P2.workc=Adult.workc or P2.workc=*)
and (P2.education=Adult.education or P2.education=*)
```

Observe that we have to join both summaries with the data set on attribute $workc$ to filter out all empty result patterns. The resulting pattern table might not be minimal and has to be re-folded over attribute $workc$ to obtain a minimal summary.

VI. EXPERIMENTS

We conducted a set of experiments on two real-life data sets. In each data set, we chose the characteristics that fit a specific use case, in order to evaluate the use and efficiency of our model in a targeted fashion.

A. Completeness summaries for sensor data

The first use case of our study considers a real sensor data set recorded by the network of our university campus. This data set includes measures about various campus resources (lighting, electricity, water, temperature, etc.). We restrict on measures pertaining to temperatures collected in 12 buildings equipped with temperature sensors and refer to this data set with **Temp**.

We build two reference data sets with different spatial coverage and over the same time interval. The first reference, noted $\mathbf{T}_{All}$, includes all spatial locations of the campus regardless of the existence of temperature sensors. The second reference, noted $\mathbf{T}_{Temp}$, restricts on localities equipped with a *temperature* sensor, that is, localities present in **Temp**. The schema of the data and the reference tables are as follows whereas their sizes are reported in Table V.

$$\mathbf{Temp}(\underline{building, floor, room, year, month, day, hour}, temp)$$

$$\mathbf{Loc_x}(\underline{building, floor, room}) \quad \mathbf{Cal}(\underline{year, month, day, hour})$$

Both reference tables $\mathbf{T}_x$ (where $x = All$ or $x = Temp$) are defined by the Cartesian product of a location table $\mathbf{Loc_x}$ and the same calendar table $\mathbf{Cal}$ and contain the key of $\mathbf{Temp}$.

TABLE V. SIZE OF REFERENCE TABLES $\mathbf{T}_{ALL}$ AND $\mathbf{T}_{TEMP}$

x	$\lvert Loc_x\rvert$	$\lvert Cal_x\rvert$	$\lvert \mathbf{T}_x\rvert = \lvert Loc_x\rvert \times \lvert Cal_x\rvert$
All	10,757	8,760	94,231,320
$Temp$	2,810	8,760	24,615,600

1) Complete and missing fragments: In addition to fragment **Temp**, we build two smaller data fragments by restricting **Temp** *spatially* to one building (building 25) and *temporally* to one month (January). The resulting fragments are respectively denoted by **Temp_OneBlg** and **Temp_OneMon**. For each data fragment ds we also define two ”narrower” reference data sets $\mathbf{T}^{ds}_{All}$ and $\mathbf{T}^{ds}_{Temp}$ obtained by using same spatial or temporal restriction of ds on reference tables $\mathbf{T}_{All}$ and $\mathbf{T}_{Temp}$.

Complete pattern summaries remain unchanged with the reference extension, but it is more efficient to restrict the study to the areas covered by sensors to avoid producing extra pattern sets (missing summary) and decrease the execution time when producing fragment summaries (see Table VI).

TABLE VI. PATTERN DERIVATION: EXECUTION TIME

data set ds	$\lvert P(ds)\rvert$	Execution time (sec)	
		T_Temp	T_All
Temp	$11,269$	$5,983$	$32,620$
Temp_OneBlg	39	45	45
Temp_OneMon	119	75	90

2) Compactness: We measure the effectiveness of patterns in terms of compactness and the efficiency of Algorithm $FoldData$. The *compactness* of a pattern table P is defined by the ratio $\lvert P\rvert/\lvert S\rvert$ between the size of summary P and the size of the data fragment S (low compactness means high compression ratio). We consider for this part the restricted reference T_{Temp} , and report results in table VII.

TABLE VII. PATTERN DERIVATION: PRELIMINARY RESULTS

data set ds	$\lvert P_C\rvert$	$\lvert Comp._C\rvert$	$\lvert P_M\rvert$	$\lvert Comp_M\rvert$
Temp	11,269	85×10^{-4}	7,086	2.86×10^{-4}
Temp_OneBlg	39	1.1×10^{-4}	36	0.28×10^{-4}
Temp_OneMon	119	13×10^{-4}	222	1.1×10^{-4}

3) Performance: In the following experiment, we evaluate the performance of algorithm $FoldData$. Table VI reports the execution time for the three data sets and both reference data sets. We notice that the execution time mainly depends on the number of generated patterns and the reference data size and the fragment size itself has a low impact on running time.

To study the evolution of the execution time w.r.t the number of patterns and the fragment size, we derive from the original data fragment **Temp**, 30 sub-fragments grouped into three categories with approximately the same completeness ratio but of different size. Figure 2 shows the running time of $FoldData$ for all fragments according to the number of generated patterns. Points of different colors denote fragments of different size ($orange = 15\%$, $violet = 10\%$ and $green = 3\%$ of the reference data set).

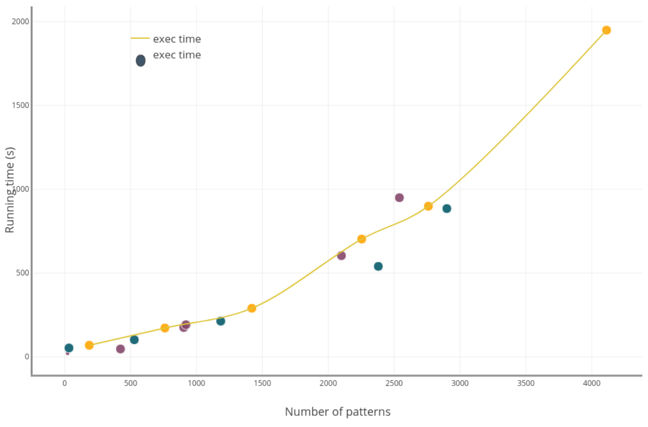


Figure 2. $FoldData$ performance

Notice that execution time is not impacted by the fragment size but grows exponentially with the number of generated patterns.

B. Census income (Adult) data set

The Adult data set is a public data set from the UCI (University of California Irvine) machine learning repository [6], which contains census data about population income. The data set consists of $32,561$ tuples over 14 attributes. For our experiments, we keep a subset of 8 attributes: one numerical attribute, *age* ranging from 17 to 90, and seven categorical attributes: *Workclass* (Private, Federal-Gov...), *Education* (Bachelors, Doctorate,...), *Marital-Status* (Married, Divorced..), *Occupation* (Tech-Support, Sales,...), *Race* (Black, White,...), *Sex* (Female, Male), *Income* (<50k,≥50k). This data set is widely used for learning population income classes (high income >50K, and low <=50k). We achieve different tasks using this data set: 1) completeness/missingness characterization regarding the occupation attribute and 2) income classes summarization.

1) Data completeness analysis: In this data set, *Occupation* and *Workclass* are the only attributes with Null values and generate the same complete and missing fragments with/without Null values. The preliminary results of the completeness analysis are shown in Table VIII.

TABLE VIII. COMPLETENESS AND MISSING PATTERNS FOR OCCUPATION/WORKCLASS

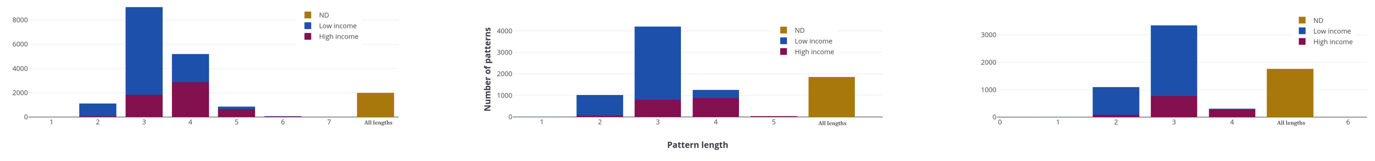
workclass/occupation	complete		missing	
data set	data	patterns	data	patterns
Adult	30718	3363	1843	521

2) Income class summarization: The results are reported in table IX. We can see that by decreasing the number of attributes, the coverage of the corresponding summaries decreases and the size of the rest increases. We also can see that the number of patterns in a summary might be higher than the data set it describes (for example, the summary for high income in $D1$). This is due to the fact that a summary precisely characterizes the fragment with respect to the whole data set and therefore contains more information about the data

TABLE IX. INCOME CLASSES SUMMARIES WITH VARIABLE ATTRIBUTES SETS

		Age : numerical						Age : categorical					
data set	Attributes	High income		Low income		Not distinguishable		High income		Low income		Not distinguishable	
		Data	Patterns	Data	Patterns	Data	Patterns	Data	Patterns	Data	Patterns	Data	Patterns
D_1	Ag,Wo,Ed,MS,Oc,Ra,Se	4382	5485	20848	10924	7544	1995	1591	1813	15227	4096	15743	1454
D_2	Ag,Ed,MS,Oc,Se	2283	1786	18164	4736	12114	1859	394	284	11235	971	20932	736
D_3	Ag,Ed,MS,Oc	1712	1106	16964	3131	13858	1762	184	133	9363	493	23014	645

Ag:age, **Wo**:Workclass, **Ed**:Education,**MS**:Marital-Status,**Oc**:Occupation,**Ra**:Race,Se:Sex



(a) Seven attributes (b) Five attributes (c) Four attributes

Figure 3. Income classes pattern summaries with variable attributes sets (Age as a numerical attribute)

fragment than the fragment itself. The table also shows that reducing the attribute Age domain, by aggregating values (numerical to categorical), lead to increasing the size of *ND*, which can be explained by the fine-grained correlation between the Age attribute and the Income. Figure 3 shows the distribution of patterns according to their length (number of constant values in a pattern). For example, in D_2 we infer that 74 patterns of length 2, are in the high-income summary, while $1,033$ belong to low-income summary. This means that for data covered by both patterns sets, we can decide about their income by knowing only two attributes among 7. We also observe the evolution of the size of the pattern set corresponding to non-distinguishable data (in yellow), increasing with attribute set restriction.

The running time increases with the number of attributes. The larger the attribute set is, the more attribute combinations have to be checked during pattern generation. Table X summarizes the running times for fragment Low in all data sets.

TABLE X. EXECUTION TIME DEPENDING ON ATTRIBUTES NUMBER

Data set	Number of attributes	Running time (s)
D_1	7	259.19
D_2	5	60.96
D_3	4	32.87

C. Compactness study for synthetic data sets

We showed in previous experiments various results for pattern summary compactness for both data sets (Temp and Adult). While the compactness for complete and missing data fragments summaries over Temp was very low, we can observe in Table IX that high/low-income fragment summaries suffer from a bad compactness, that even exceeds 1 in some cases (D_1: Age numerical : High income). This difference can be explained by the data distribution over the fragments: sensors fail in a continuous time intervals, leading to long complete and incomplete data sequences, which can be summarized by a small number of generic patterns. On the other hand, high and low-income tuples in the Adult data set are distributed in a

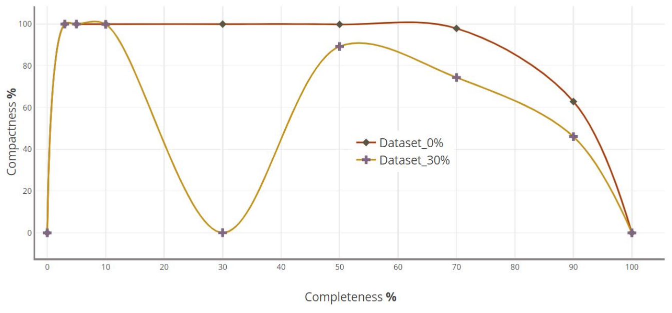


Figure 4. Random evolution

random way over the attributes domains, which leads to large sets of specific patterns. To better understand this phenomena, we created two series of synthetic data sets starting from the **Temp** data set and simulated a set of sensors producing data with different missing/available data distributions.

1) Dense distribution data sets are obtained by sequentially (in time order) adding new measures to T_{Temp}.
2) Sparse distribution data sets are generated by randomly deleting measures from T_{Temp}.

We generate a series of data sets by increasing and decreasing the completeness of three initial data sets with completeness fixed to 0% (empty), 30% and 50% respectively. For each initial data set, we simulate two types of evolution (1) by successively inserting tuples from the reference until reaching full completeness and (2) by successively deleting tuples until reaching emptiness. The insertion and deletions follow two strategies: i) a sequential strategy that selects the (inserted or deleted) tuples using their spatial and temporal domain order preserving the original data distribution and which we call *sequential evolution*, and ii) a random strategy that picks these tuples in a random fashion, we call *random evolution*.

Figures 4 and 5 depict the variation of compactness for each data set and its evolution. In the randomly evolving data

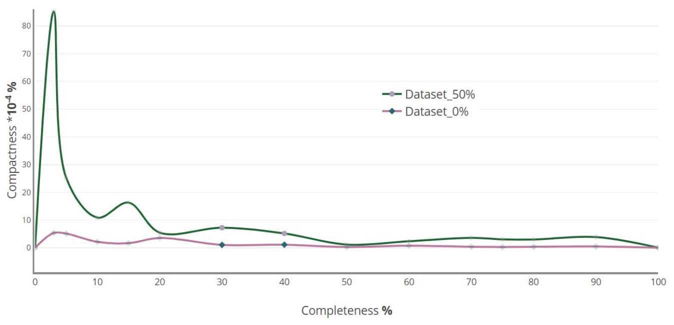


Figure 5. Sequential evolution

sets (Figures 4), the compactness of a random data set with 30% completeness evolves symmetrically in both directions (insertion and deletion). Random insertions and deletions first generate new patterns and cause at some point the fusion of fine-grained patterns to coarser-grained ones to achieve maximum compactness at both extremities. In the sequentially evolving data sets, we observe the same trend with a lower amplitude for a data set with 50% initial completeness: insertions lead to a faster completion of the partial partitions (thanks to order sensitive updates) and thus to faster derivation of coarser patterns without deriving all their subsumed patterns.

VII. Conclusion

We have proposed a formal summarization model and introduced reasoning mechanisms for characterizing the contents of data fragments relative to a complete dataset. We illustrated the use of our framework within two application scenarios for reasoning about information completeness and for characterizing fragment summaries. We have illustrated our approach and validated its implementation experimentally on two data sets. A natural extension under study is the use of Apache Spark [19] for computing and querying summaries for very large fragmented data sets. We also intend to implement a visual query interface for the interactive exploration of fragment summaries.

References

[1] W. A. Voglozin, G. Raschia, L. Ughetto, and N. Mouaddib, "Querying a summary of database," Journal of Intelligent Information Systems, vol. 26, no. 1, Jan. 2006, pp. 59–73.

[2] F. Buccafurri, F. Furfaro, D. Sacca, and C. Sirangelo, "A Quad-tree Based Multiresolution Approach for Two-dimensional Summary Data," in Proceedings of the 15th International Conference on Scientific and Statistical Database Management, ser. SSDBM '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 127–140.

[3] A. Cuzzocrea and D. Sacc, "H-IQTS: A Semantics-aware Histogram for Compressing Categorical OLAP Data," in Proceedings of the 2008 International Symposium on Database Engineering & Applications, ser. IDEAS '08. New York, NY, USA: ACM, 2008, pp. 209–217.

[4] H.-J. Lenz and A. Shoshani, "Summarizability in OLAP and statistical data bases," in Scientific and Statistical Database Management, 1997. Proceedings., Ninth International Conference on. IEEE, 1997, pp. 132–143.

[5] L. V. Lakshmanan, J. Pei, and J. Han, "Quotient cube: How to summarize the semantics of a data cube," in Proceedings of the 28th international conference on Very Large Data Bases. VLDB Endowment, 2002, pp. 778–789.

[6] "Adult income dataset," Accessed on March 2019, https://archive.ics.uci.edu/ml/index.php.

[7] J. Chen, J.-Y. Pan, C. Faloutsos, and S. Papadimitriou, "TSum: fast, principled table summarization," in Proceedings of the Seventh International Workshop on Data Mining for Online Advertising - ADKDD '13. Chicago, Illinois: ACM Press, 2013, pp. 1–9.

[8] G. Raschia and N. Mouaddib, "SAINTETIQ: a fuzzy set-based approach to database summarization," Fuzzy sets and systems, vol. 129, no. 2, 2002, pp. 137–162.

[9] R. Saint-Paul, G. Raschia, and N. Mouaddib, "General Purpose Database Summarization," in Proceedings of the 31st International Conference on Very Large Data Bases, ser. VLDB '05. Trondheim, Norway: VLDB Endowment, 2005, pp. 733–744.

[10] M.-L. Lo, K.-L. Wu, and P. S. Yu, "Tabsum: A flexible and dynamic table summarization approach," in Distributed Computing Systems, 2000. Proceedings. 20th International Conference on. IEEE, 2000, pp. 628–635.

[11] M. van Leeuwen and J. Vreeken, "Mining and Using Sets of Patterns through Compression," in Frequent Pattern Mining, C. C. Aggarwal and J. Han, Eds. Cham: Springer International Publishing, 2014, pp. 165–198.

[12] A. Koopman and A. Siebes, "Characteristic Relational Patterns," in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser. KDD '09. New York, NY, USA: ACM, 2009, pp. 437–446.

[13] T. Imieliński and W. Lipski, "Incomplete information in relational databases," in Readings in Artificial Intelligence and Databases. Elsevier, 1988, pp. 342–360.

[14] A. Motro, "Integrity = Validity + Completeness," ACM Trans. Database Syst., vol. 14, no. 4, Dec. 1989, pp. 480–502.

[15] A. Y. Levy, "Obtaining Complete Answers from Incomplete Databases," in Proceedings of the 22th International Conference on Very Large Data Bases, ser. VLDB '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 402–412.

[16] S. Razniewski, F. Korn, W. Nutt, and D. Srivastava, "Identifying the extent of completeness of query answers over partially complete databases," in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4 2015, pp. 561–576.

[17] W. Fan and F. Geerts, "Relative Information Completeness," ACM Trans. Database Syst., vol. 35, no. 4, Oct. 2010, pp. 27:1–27:44.

[18] B. Sundarmurthy, P. Koutris, W. Lang, J. F. Naughton, and V. Tannen, "m-tables: Representing missing data," in 20th International Conference on Database Theory, ICDT, Venice, Italy, March 2017, pp. 21:1–21:20.

[19] M. Zaharia et al., "Apache spark: a unified engine for big data processing," Communications of the ACM, vol. 59, no. 11, 2016, pp. 56–65.