Query-Oriented Answer Imputation for Aggregate Queries

Fatma-Zohra Hannou, Bernd Amann(✉), and Mohamed-Amine Baazizi

Sorbonne Université, CNRS, LIP6, Paris, France
{fatma.hannou,bernd.amann,mohamed-amine.baazizi}@lip6.fr

Abstract. Data imputation is a well-known technique for repairing missing data values but can incur a prohibitive cost when applied to large data sets. Query-driven imputation offers a better alternative as it allows for fixing only the data that is relevant for a query. We adopt a rule-based query rewriting technique for imputing the answers of analytic queries that are missing or suffer from incorrectness due to data incompleteness. We present a novel query rewriting mechanism that is guided by partition patterns which are compact representations of complete and missing data partitions. Our solution strives to infer the largest possible set of missing answers while improving the precision of incorrect ones.

Keywords: Data imputation · Aggregation queries · Partition patterns

1 Introduction

Data incompleteness naturally leads to query results of poor quality and repairing missing data is a common data cleansing task. Data imputation consists in estimating missing data values manually or by using statistical or predictive models built from existing available data. Repairing large data sets can lead to important imputation costs which may be disproportional to the cost of certain query workloads. In particular for analytic queries, imputation at the raw data level might be inefficient for repairing aggregated query results. Query-driven imputation allows for rectifying only the values that are relevant for the evaluation of a query and, hence, can significantly reduce the amount of data to be processed. Although this approach is appealing, it has received little attention within the data cleansing community. A notable exception is ImputeDB [2] where data imputation is performed during query execution by injecting statistical estimators into the query plans. While this approach achieves its primary goal of repairing only the data accessed by the queries, it requires the extension of existing query processors to accommodate for the new imputation operators. In this article, we are interested in the imputation of aggregated query results where partition-wise imputation at a higher granularity might be preferred to the tuple-wise imputation at the raw data level. Our approach is based on the assumption that incorrect values obtained by incomplete partitions are of lower quality than values obtained by imputation. Instead of instrumenting the query execution plan, we adopt a declarative approach where experts express domain and data specific imputation rules which identify missing or incorrect query answers and estimate their values by aggregating available correct results. This approach allows data analysts to capture the semantics of the underlying data imputation process and facilitates the interpretation of the imputation results.

As an example, consider the following SQL query Q_{kwh} over some table $Energy(B, F, R, W, D, kWh)$ storing daily kWh measures for day D of week W and rooms R in floor F of building B. Query Q_{kWh} computes the average weekly energy consumption for three floors in building 25:

select B, F, W, sum(kWh) **as** kWh
from Energy
where B = 25 **and** F **in** (1,2,3)
group by B, F, W

Table *Energy* might miss some measures for these floors. This incompleteness can be the result of tuples containing *null* values in attribute kWh or defined with respect to some reference tables (maps, calendars etc.). In both cases, a first step consists in identify missing data tuples, and subsequently trace how they impact the quality of query Q_{kWh}. Table 1 shows respectively a representation of table *Energy* (ordered by week and floor), and the query answer $Q_{kwh}(Energy)$. Missing measures are indicated by *null* values. For example, for week 1, *Energy* contains all measures of floors 1 and 3 (we consider that each floor has only one room and a week has three days) and misses one measure for floor 2. It misses all measures for floor 1 and week 2. Each tuple in the query answer is obtained by aggregating a partition of the input data and annotated `correct` if

Table 1. *Energy* table and $Q_{kwh}(Energy)$

Energy

B	**F**	R	**W**	D	kWh
25	1	1	1	1	12.3
25	1	1	1	2	10.1
25	1	1	1	3	9.6
25	2	1	1	1	8.3
25	2	1	1	2	6.4
25	2	1	1	3	null
25	3	1	1	1	5.3
25	3	1	1	2	7.2
25	3	1	1	3	6.1
25	1	1	2	1	null
25	1	1	2	2	null
25	1	1	2	3	null
...	...	...	...	...	...

$Q_{kWh}(Energy)$

B	**F**	**W**	**kWh**	**label**
25	1	1	32.0	`correct`
25	2	1	14.7	`incorrect`
25	3	1	18.6	`correct`
25	1	2	null	`missing`
...	...	...	...	...

the partition is complete, `missing` if the partition is empty and `incorrect` if the partition is not empty by incomplete. Data imputation is a common approach for fixing incomplete data by applying statistical or logical inference methods. Most approaches apply imputation independently of the query at the raw data level. As shown in [2], repairing the entire data may turn expensive when data is large and inefficient when the query only needs to fix a subset of values to improve the query result quality. A second argument we develop in this article is that, for aggregation queries, it might have more sense to estimate missing or incorrect results at the granularity of the query result itself by exploiting specific knowledge about the input data sets at the aggregation level. For this, we propose rule based approach for repairing dirty data similar to [5] where domain experts can define imputation rules using correct available observation to repair missing and incorrect results. For example, an expert can state by the following imputation rule that some missing or incorrect query result (due to incomplete data) for a given floor could be repaired by taking the average of all available correct *results*

$$r_0 : (B : x, F : \_, W : y) \leftarrow (B : x, F : \_, W : y), avg(kWh)$$

More precisely, any missing or incorrect measure for some floor in building x and for week y that matches with the left-hand side of the rule will be estimated by the average over all correct measures for the same building and week (tuples matching the right-hand side of the rule).

Our imputation process is based on a pattern representation of queries and data for identifying correct, incorrect and missing query answers and for selecting and evaluating imputation rules. In summary, the main steps of our imputation approach are the following:

1. experts define imputation rules for repairing aggregate query results using information about correct observations;
2. the system detects and summarizes correct, incomplete and missing query answers in form of partition patterns;
3. the system selects for each missing or incorrect partition pattern one or several rules for imputing data;
4. the selected imputation rules are translated into standard SQL queries that generate correct results for missing or incorrect query answers.

Contribution and Paper Outline: Our imputation model for summarizing, analyzing and repairing aggregate query answers is presented in Sect. 2. Related work is presented in Sect. 3. We describe the imputation process in Sect. 4 and validate it experimentally in Sect. 5. Conclusion and future work directions are presented in Sect. 6.

2 Imputation Model

Our imputation model is defined for a particular sub-class of SQL aggregate queries:

Definition 1. *Let Q be valid SQL aggregate query of the form*

select *S, agg(m)* ***from*** *T* ***where*** *P* ***group by*** *G*

where condition P only uses equality predicates with constants.

Without loss of generality, we assume that P is in disjunctive normal form. For example, query Q_{kWh} can be rewritten by transforming the **where** clause into (B = 25 **and** F = 1)**or** (B = 25 **and** F = 2)**or** (B = 25 **and** F = 3).

We now introduce the notion of query pattern for representing the partitions generated by the **group by** clause of an SQL aggregate query.

Definition 2. *A* query pattern *is a tuple $q = (a_1 : x_1, ..., a_n : x_n)$ where for each attribute a_i, its values $x_i \in dom(a_i) \cup V \cup \{*\}$ is (1) a constant in the domain of attribute a_i or (2) a distinct variable in a set of variables V or (3) the wildcard symbol $*$.*

For example, $(B : 25, F : x, R : *)$ is a query pattern where $x \in V$ is a variable.

Definition 3. *Let Q be some valid SQL aggregate query as defined in Definition 1 and A be a key of the input table containing all attributes in Q except the aggregated attribute. We can then define a set of query patterns $\mathcal{Q}$ over A which contains a query pattern $q_i \in \mathcal{Q}$ for each disjoint d_i in the* ***where*** *clause such that (1) all attributes A_j in d_i are represented by the corresponding constants c_j in q_i, (2) all other attributes in the* ***group by*** *clause are distinct variable attributes and (5) all attributes in A and not in Q are wildcard attributes.*

For example, suppose that (B, F, R, W, D) is a key of table *Energy* (see Sect. 1). Then, the SQL query Q_{kWh} generates the query pattern set $\mathcal{Q} = \{(B : 25, F : 1, R : *, W : \_, D : *), (B : 25, F : 2, R : *, W : \_, D : *), (B : 25, F : 3, R : *, W : \_, D : *)\}$. Observe that all query patterns of a query share the same wildcard attributes and if q does not contain any wildcard attributes, then the corresponding SQL query corresponds to a simple conjunctive query which returns the measure values of the matching tuples (without aggregation and group-by clause).

The instance of a query pattern q defines a subset of the partitions generated by the **group by** clause over the tuples filtered by the **where** clause of the corresponding SQL query. This filtered partitioning can formally be defined by a partial equivalence relation over the query input tuples:

Definition 4. *A tuple t* matches *a query pattern q, denoted $match(t, q)$, if $t.a_i = q.a_i$ for all* constant *attributes in q. Two tuples t and t' matching some query pattern q are* equivalent *in q, denoted $t \equiv_q t'$, if $t.a_j = t'.a_j$ for all* variable *attributes a_j in q (t and t' only can differ for wildcard attributes).*

A pattern p defines for each matching tuple t an equivalence class $\Phi_q(t) = \{t'|t \equiv_p t'\}$.

Definition 5. *The instance of a query pattern q in some table M, denoted $I(M, q)$, contains all equivalence classes (partitions) of tuples in M.*

It is easy to see that (1) when p does not contain any wildcard attribute, then $I(M, q) = \{\{t\}|t \in I(q, M)\}$ contains a singleton for each matching tuple in M, and (2) when q does not contain any variable, $I(q, M) = \{\Phi_q\}$ contains a single partition $\Phi_q \subseteq M$. For example, the equivalence class $\Phi_q(t)$ of tuple $t = (B : 25, F : 1, R : 1, W : 2, D : 3)$ defined by pattern $q = (B : 25, F : 1, R : *, W : x_w, D : *)$ contains all tuples of building 25, floor 1 and week 2. The equivalence class $\Phi_q(t')$ with the same pattern q for tuple $t' = (B : 26, F : 1, R : 1, W : 2, D : 3)$ is empty. Finally, $q' = (B : 25, F : 1, R : *, W : 2, D : *)$ defines a unique equivalence class for all tuples of floor 1 in building 25 and week 2.

Constrained Tables and Partition Patterns: We follow the approach of *relative information completeness* [7] for modeling the completeness of a data table M.

Definition 6. *Let M be a relational table and A be a key in M. Table R is called a* reference table *for* data table M *if R contains all keys of M: $\pi_A(M) \subseteq R$. The pair $T = (M, R)$ is called a* constrained table.

Observe that any table $M'(\underline{A}, val)$ with key A and *with null values for attribute val* can be decomposed into an equivalent constrained table $T = (M, R)$ where measure table $M = \sigma_{val\,is\,not\,null}(M')$ contains all tuples in M *without null values* and $R = \pi_A(M')$ contains all key values in M'. At the opposite, we can build from any constrained table $T = (M, R)$ a relational table $S = R \ltimes M$ with *null* values (left outer join of R with M). For example, table *Energy* has key (building, floor, room, week, day) and we can decompose *Energy* into a measure table M defined by: **select** $*$ **from** Energy **where** kWh **is not null** and a reference table R defined by **select** B, F, R, W, D **from** Energy.

In order to reason about the completeness of data *partitions*, we introduce the concept of *partition pattern* [14].

Definition 7. *Let $T = (M, R)$ be a constrained table with reference attributes A. A partition pattern p is a query pattern without variables over attributes A. A partition pattern p is*

- complete in table T *if its instance (partition) in R is equal to its instance (partition) in M: $I(p, R) = I(p, M)$.*
- incomplete in table T *if its instance (partition) in T is strictly included in its instance (partition) in R: $I(p, M) \subset I(p, R)$.*
- missing in table T *if it is incomplete and its instance in M is empty: $I(p, R) \neq \emptyset \wedge I(p, M) = \emptyset$.*

For example, table *Energy* in Table 1 contains all measures for (1) all rooms and days of the first week and the first floor and (2) misses all measures for floor 1 and week 2. These observations are respectively summarized by pattern pattern c_1 in table $\mathcal{C}$ and e_1 in in table $\mathcal{E}$ (Table 2). More exactly, pattern tables $\mathcal{C}$ and $\mathcal{E}$ define the complete and missing partitions in the subset of keys M identifying all non-null tuples in *Energy* with respect to the reference table R, which contains all keys of *Energy*.

Both pattern tables are covering and minimal in the sense that they contain respectively all *complete* and all *empty* patterns of T and no pattern that is subsumed by another pattern in the same table. We call $\mathcal{C}$ and $\mathcal{E}$ the complete and empty pattern (fragment) summaries of T and an algorithm for generating these summaries is described in [11].

Imputation rules and imputation queries: Imputation rules repair the results of aggregation queries by estimating missing tuple values. They are defined using *query patterns* characterizing the tuples that should be repaired and the tuples that should be used for their reparation.

Table 2. Complete and missing pattern tables for *Energy* data table

$\mathcal{C}$	**B**	**F**	R	**W**	D
c_1	25	1	*	1	*
c_2	25	2	*	1	*
c_3	25	2	*	2	*
c_4	25	3	*	1	*
c_5	25	3	2	2	*
c_6	25	5	*	1	*
c_7	25	5	*	2	*
c_8	26	*	*	*	*

$\mathcal{E}$	**B**	**F**	R	**W**	D
e_1	25	1	*	2	*
e_2	25	*	*	3	*
e_3	25	3	1	2	*
e_4	25	4	*	*	*

Definition 8. *An* imputation rule *for some set of reference attributes A and some measure attribute val is an expression of the form $r : q_m \leftarrow q_a, imp$ where (1) q_m and q_a are query patterns over A* without wildcards*, (2) all variables shared by q_m and q_a are bound to the same attribute in q_m and q_a and (3)* imputation expression *imp is an aggregation function transforming a set of values in the domain of m into a single value in the domain of m.*

In the following we use the anonymous variable _ for denoting non-shared variables. For example, we can define the following imputation rules for missing kWh values:

$$r_1 : (B:x, F:\_, W:y) \leftarrow (B:x, F:\_, W:y), (max(kWh) + min(kWh))/2$$
$$r_2 : (B:x, F:y, W:3) \leftarrow (B:x, F:y, W:2), kWh$$
$$r_3 : (B:\_, F:4, W:x) \leftarrow (B:\_, F:4, W:x), avg(kWh)$$

Imputation rule r_1 produces an estimation of the weekly electricity consumption of some floor in some building by the midrange of all *correct* consumption values over *other* floors of the same building and the same week. Rule r_2 takes the correct consumption of a floor in week 2 for estimating the value of the same floor at week 3 (the aggregation function returns the value of the generated singleton). Finally, rule r_3 takes the average of all correct values for floor 4 in all buildings to repair the value of the same floor in some building for the same week.

The formal semantics of an imputation rule is defined with respect to a query Q, a table *ImputeD* of result tuples to be repaired by the rule and a table *AvailableD* of all correct values which can be used for imputation. Observe that table *ImputeD* contains all results generated by incomplete partitions and all missing results corresponding to empty partitions whereas table *AvailableD* contains all possible correct tuples that are returned by a new query Q' which is obtained by removing the **where** clause of Q (imputation rules can use all correct results of Q but also other correct values that are generated by the same aggregation function over complete partitions).

Definition 9. *Let ImputeD and AvailableD be two tables that contain the tuples to be repaired and the tuples which can be used for reparation. Then, the semantics of an imputation rule* $r : q_m \leftarrow q_a, imp$ *is defined by the following* imputation query $Q(r)$ *over ImputeD and AvailableD where A contains all attributes in* q_m *and S is the set of variable attributes shared by* q_m *and* q_a*:*

select *x.A, imp* ***as*** *m*
from *ImputeD x, AvailableD y*
where $match(x, q_m)$ ***and*** $match(y, q_a)$ ***and*** $x.S = y.S$
group by *x.A*

The previous imputation query joins all tuples $x \in ImputeD$ matching q_m with the set of tuples y in *AvailableD* matching q_a over the shared attributes S, partitions the obtained table over all rule attributes and finally applies the imputation expression *imp* to estimate a value for m. For example, the previous imputation rule r_2 can be rewritten into the following SQL query :

select x.B, x.F, x.W, y.kWh
from *ImputeD* x, *AvailableD* y
where x.W = 3 **and** y.W = 2 **and** x.B = y.B **and** x.F = y.F ;

where *AvailableD* corresponds to the correct results generated by query Q_{kWh} without its filtering condition. We will show in Sect. 4 how to exploit partition patterns for identifying missing and incorrect query results and evaluating imputation rules.

3 Related Work

Missing data is an ancient problem in statistical analysis [1] and data imputation has been studied for many decades by the statisticians [1] and more recently by the database community [3]. Different imputation approaches have been developed including Mean Imputation [9], which is still one of the most effective techniques, K-Nearest Neighbor [9] and Hot Deck Imputation [18] (k=1) aggregating the k most similar available samples, Clustering-Based Imputation [12] estimating missing values by the nearest neighbor in the same cluster built over the non-missing values. More recently, Learning-Based Imputation models adapt machine learning techniques and represent data imputation as a classification and regression problem [15]. Crowdsourcing techniques are also used for achieving data completion or repairing tasks and achieve high-quality results balanced by an increased cost of human effort. Rule based cleaning is a middle-ground solution that allows referring to human for drawing cleaning rules, but automates the reparation process. A notable example is the work of [5], introducing a user interface for editing cleaning rules, and running automatic cleaning tasks. The solution presented in [13] allies an SQL-like languages and machine learning techniques to offer non expert users an interface to achieve data imputation. Several classes of data integrity constraints also have been applied for detecting and repairing missing or incorrect data [6]. For example, [8] proposes a formal model based on master data to detect missing and incorrect data and editing rules for repairing this data. The more recent work of [17] introduces the notion of fixing rules to capture semantic errors for specific domains and specify how to fix it. Our work adopts the same idea of using rules for repairing aggregate query answers using a different approach for identifying missing and incorrect results generated by empty or incomplete data partitions.

For large datasets, data imputation can become very expensive and inefficient. In contrast, query result estimation techniques consider repairing query answers, obtained by applying queries to incomplete (or incorrect) data. Sampling methods have been first used for run-time optimization through approximate queries [10,16] over representative sample instead of the entire dataset. In the same spirit, [16] integrates sampling techniques together with a cleaning strategy to optimize during run-time. The price to pay for the reduced time and cleaning effort is a bias introduced by the cleaning and sampling. [4] tackles the problem in the absence of any knowledge about the extent of missing tuples and estimates the impact of unknown unknowns on query results. The work introduces statistical models for quantifying the error in query results due to missing tuples. The goal of dynamic imputation [2] is to incorporate the missing data imputation into the query optimization engine. Physical query plans are augmented with two additional operators, that delete tuples with missing values and replace them with new values. Only data involved in the query evaluation are imputed, and the replacement is achieved at different query execution steps.

4 Query Imputation Process

Our imputation process is decomposed into four steps. The first step consists in identifying the set of all partition patterns $ImputeP(Q)$ summarizing the partitions to be repaired and the set of partition patterns $AvailableP(Q)$ of partitions that can be used for reparation. Step 2 consists in identifying the set of all rules that can be used for repairing $ImputeP(Q)$ by using $AvailableP(Q)$.

In this step, a rule is chosen if and only if it can repair at least one answer tuple and if there exists at least one correct value that can be used for imputation. The result of this step is a set of *candidate imputations*. The third step consists in creating a *sequence of candidate imputations* which repair the missing and incorrect tuples. Observe that a tuple might be repaired by several queries and we assume that each imputation query overwrites conflicting repaired tuples generated by the previous queries. Finally, step 4 consists in generating the imputation queries following the imputation strategy of step 2.

Step 1: Identify correct, incorrect and missing result tuples: For identifying correct, incorrect and missing answers, we first extend the notion of pattern matching from tuples to query patterns.

Definition 10. *A query pattern q* matches *a partition pattern p, denoted by $match(q,p)$, if for all* constant *attributes $q.a_i$ in q, $q.a_i = p.a_i$ or $p.a_i = *$. If $match(q,p)$, we can define a mapping ν from the variable attributes a_i in q to the attributes in p such that $\nu(q.a_i) = p.a_i$. Then, a query pattern q (1)* fully matches *partition pattern p, denoted by $full(q,p)$, if $\nu(q)$ matches p and (2) partially matches p, denoted by $partial(q,p)$, otherwise. Partition pattern $\nu(p)$ is called the* matching pattern *of q for p.*

For example, query pattern $q = (25, \_, *, \_, *)$ matches all patterns in $\mathcal{C}$ and $\mathcal{E}$ except pattern $c_8 : (26, *, *, *, *)$[1]. It fully matches patterns $c_1 : (25, 1, *, 1, *)$, $c_2 : (25, 2, *, 1, *)$, $c_3 : (25, 2, *, 2, *)$, $c_5 : (25, 3, 2, 2, *)$, $c_6 : (25, 5, *, 1, *)$, $c_7 : (25, 5, *, 2, *)$, $e_1 : (25, 1, *, 2, *)$ and $e_2 : (25, *, *, 3, *)$ and partially matches pattern $e_3 : (25, 3, 1, 2, *)$ and $c_4 : (25, 3, *, 1, *)$.

Let P be a set of partition patterns and $\mathcal{Q}$ be a set of query patterns of some query Q. Then we denote by $match(P, \mathcal{Q})$, $partial(P, \mathcal{Q})$, and $full(P, \mathcal{Q})$ the sets of partition patterns in P that are matched, partially matched and fully matched by query patterns $q \in \mathcal{Q}$. By definition, $partial(P, \mathcal{Q}) = match(P, \mathcal{Q}) - full(P, \mathcal{Q})$.

Definition 11. *Let $W(Q)$ be the set of wildcard attributes in the query patterns $\mathcal{Q}$ of some query Q and q be a query pattern over all variable and constant attributes in the query patterns of Q. Then we denote by q^* the query pattern where all attributes in W are wildcard attributes. Pattern q^* is called the extension of q in Q.*

The extension of query pattern $q = (B : 25, F : 2, W : \_)$ is $q^* = (B : 25, F : 2, R : *, W : \_, D : *)$ and the extension of tuple $t = (B : 25, F : 1, W : 1)$ in Q is pattern $t^* = (B : 25, F : 1, R : *, W : 1, D : *)$.

Proposition 1. *Given a query Q over some constrained table $T = (M, R)$ with complete pattern summary $\mathcal{C}$ and missing pattern summary $\mathcal{E}$. Let $\mathcal{Q}$ be the query pattern set of Q. Then, for any tuple t in the reference table of Q the following holds:*

[1] We omit attribute names when they're not necessary for understanding.

- *t is in the result of Q and* correct *iff t^* matches a pattern $p \in full(\mathcal{C}, \mathcal{Q})$;*
- *t is in the result of Q and* incorrect *iff t^* matches a pattern $p \in partial(\mathcal{E}, \mathcal{Q})$ (or, equivalently $p \in partial(\mathcal{C}, \mathcal{Q})$;*
- *t is* missing *in the result of Q iff a pattern $p \in full(\mathcal{E}, \mathcal{Q})$ matches t^*.*

Since query pattern $q = (25, 1, *, \_, *)$ fully matches patterns c_1, all answer tuples t of Q_{kWh} where t^* matches c_1 are correct. For example, for tuple $t = (25, 1, 1)$, its extension $t^* = (25, 1, *, 1, *)$ matches $c_1 : (25, 1, *, 1, *)$. The same argument holds for pattern $q' = (25, 2, *, \_, *)$ and tuples $(25, 2, 1)$ and $(25, 2, 2)$ On the opposite, since q also fully matches patterns $e_1 : (25, 1, *, 2, *)$ and $e_2 : (25, *, *, 3, *)$, all answer extensions matching these patterns are missing in the result. Finally, $q'' = (25, 3, *, \_, *)$ partially matches pattern $e_3 : (25, 3, 1, 2, *)$ (or, equivalently, pattern $c_4 : (25, 3, *, 1, *)$), all extended answer patterns matched by these patterns, like for example $(25, 3, 1)^* = (25, 3, *, 1, *)$, are incorrect.

By Proposition 1 and Definition 10, the set of missing or incorrect tuples exactly corresponds to the set of the corresponding extended tuples matching $\mathcal{E}$.

Definition 12. *Given query Q over some table $T = (M, R)$ with pattern tables $\mathcal{C}$ and $\mathcal{E}$ and query pattern set $\mathcal{Q}$. We can then define the following sets of patterns for Q:*

- *$ImputeP(Q) = full(\mathcal{E}, \mathcal{Q}) \cup partial(\mathcal{E}, \mathcal{Q}) = full(\mathcal{E}, \mathcal{Q})$*
- *$AvailableP(Q) = \{p | p \in \mathcal{C} \wedge \forall A \in W(Q) : p.A = *\}$*

$ImputeP(Q)$ contains all patterns describing incomplete or missing partitions (to be repaired) in the result of Q whereas $AvailableP(Q)$ describes all complete partitions that can be used for repairing Q. In the following step, we explain how we can use these two sets for filtering imputation rules for some aggregation query Q.

Step 2: Generate Candidate Imputations: Missing and incorrect answers of some aggregation query Q (query pattern set $\mathcal{Q}$) are estimated by imputation queries. Each imputation query is generated by an imputation rule and repairs some missing and incorrect tuples. We assume that the complete and missing data partitions are represented by a complete and missing pattern summary as defined before. We first define the notion of candidate imputation.

Definition 13. *Let $ImputeP(Q)$ be the imputation pattern set and $AvailableP(Q)$ the reparation pattern set of Q. A* rewriting *ω for $p_m \in ImputeP(Q)$ is an expression $\omega : p_m \leftarrow^r P_a$ where there exists an imputation rule $r : q_m \leftarrow q_a, f_{imp}$ such that the extended query pattern q^*_m matches p_m with ν and $P_a \subseteq \mathcal{C}$ is a non-empty set of complete patterns in $\mathcal{C}$ that are matched by $\nu(q^*_a)$.*

We say that rule r generates rewriting ω and call $\nu(q^*_m)$ the *imputation pattern of* ω and $\nu(q^*_a)$ the *repair pattern* of ω. All rules r where there exists at least one rewriting are called *candidate imputations* for Q. For example, $\omega_1 : e_1 \leftarrow^{r_1} \{c_3, c_7\}$ is a candidate imputation for $e_1 : (25, 1, *, 2, *)$ generated by rule r_1 with imputation pattern $\nu(q^*_m) = e_1 : (25, 1, *, 2, *)$, repair pattern $\nu(q^*_a) = (25, \_, *, 2, *)$ and $P_a = \{c_3 : (25, 2, *, 2, *), c_7 : (25, 5, *, 2, *)\}$, Similarly, $\omega_2 : e_2 \leftarrow^{r_2} \{c_3, c_7\}$ is a candidate imputation for $e_2 : (25, *, *, 3, *)$ using rule r_2 with imputation pattern $\nu(q^*_m) = e_2 : (25, *, *, 3, *)$, repair pattern $\nu(q^*_m) = (25, *, *, 2, *)$ and $P_a = \{c_3 : (25, 2, *, 2, *), c_7 : (25, 5, *, 2, *)\}$ and Finally, $\omega_3 : e_2 \leftarrow^{r_3} \{c_8\}$ is second a candidate imputation for $e_2 : (25, *, *, 3, *)$ using rule r_3 with imputation pattern $\nu(q^*_m) = e_2 : (25, *, *, 3, *)$, repair pattern $\nu(q^*_a) = (\_, 4, *, 2, *)$ and $P_a = \{c_8 : (26, *, *, *, *)\}$.

Step 3: Imputation strategy: The result of step 2 is a set of candidate imputation rules where there exists at least one rewriting. Given a set of candidate imputations $\mathcal{R}$ for some aggregation query Q, the goal is to define an ordered sequence of candidate imputations for repairing the answer of Q. This sequence is called an *imputation strategy.* The goal of a strategy is to solve two kinds of conflicts. First there might exist several candidate imputations for the same partition pattern $p_m \in ImputeP(Q)$ as shown in the example above for pattern e_2. Second, patterns in $ImputeP(Q)$ might not be disjoint and repair a subset of shared tuples. For example, missing patterns $e_2 : (25, *, *, 3, *)$ and $e_4 : (25, 4, *, *, *)$ might share the partition $(25, 4, 3)$. A standard way for solving such conflicts is to apply a multiple-imputation strategy which consists in applying all candidate imputations and combining the estimated results through some statistical methods. In this article, we adopt a different strategy which consists in regrouping all candidate imputations for each rule and evaluating these imputation groups following a static priority order defined over the imputation rules. We can show that this process is deterministic since each imputation rule generates at most one imputation value for any missing tuple.

Imputation rules can be ordered in different ways. For example, one might prefer "specialized" rules to more "generic" rules where specialization can be expressed by the number of constants in the and shared variables. For example, rule r_3 is then considered more specialized than rule r_1 since it contains more constants whereas rule r_2 is more specialized than r_3 since it contains more shared variables (with the same number of constants). Another strategy is to order the rules using some statistical estimations about data distribution, bias and completeness or domain specific expert knowledge about the system generating the data. For example, if the kWh values for floor 4 are quite similar over all buildings for a given week, rule r_d might be preferable to rule r_c. Rule r_1 might be preferred to the other rules if the kWh values do not vary over the floors of the same building.

Step 4: Imputation query generation: As shown in Definition 9, each candidate imputation $r : q_m \leftarrow q_a, f_{imp}$ generates an imputation query joining the table *ImputeD* of values to be repaired with the table *AvailableD* containing all correct values. As explained in Sect. 2, table *AvailableD* is shared by all imputation queries, and can be obtained by removing the filter condition (where clause) of query Q and matching the result with the pattern table $AvailableP(Q)$ (see Definition 12). For performance reasons we precompute this table once and store the result, and reuse it for all imputation queries. Table $ImputeD$ can be obtained by matching the result Q with pattern table $ImputeP(Q)$. Each rule $r : q_m \leftarrow q_a, f_{imp}$ then generates the following imputation query over tables $ImputeP(Q)$, the result table $Result$ of Q and $AvailableD(Q)$ where S is the set of variable attributes shared by q_m and q_a and A is the set of remaining attributes in q_m:

select x.A, x.S, f_{imp} **as** m
from $ImputeP(Q)$ p, Result x, $AvailableD$ y
where match(x,q_m) **and** match(y,q_a) **and** x.S = y.S **and** match(x,p)
group by x.A, x.S

In the experiments we use a variation of imputation queries which returns the pattern cover, for partitions to be repaired. This is more efficient since partitions covered by the same pattern are imputed with the same value. An example of such a rewriting is shown in the expriments.

5 Experiments

In this section we investigate the effectiveness and efficiency of our pattern-based approach for repairing analytic queries answers. We consider a real dataset of temperature measure collected by the sensor network at our university campus. The data table $Temp(building, floor, room, year, month, day, hour, value)$ used in our study contains temperatures collected in 12 buildings during one year. $Temp$ features both spatial and temporal incompleteness since sensors only partially cover the campus buildings and operate erratically. In addition to the temperature measures, we consider a second data set $Occ(building, floor, room, occupation, area)$ that records campus rooms areas and occupations. Complete and empty pattern summaries are computed by a pattern generation algorithm described in [11]. This algorithm produces pattern summaries with respect to the campus map table and the calendar. Data and pattern tables cardinalities are reported in Table 3. For example data table D_{Temp} contains $1,321,686$ tuples for a reference table R_{Temp} of $24,615,600$ tuples ($Temp$ only covers about 5% of reference R_{Temp}), and generates $11,268$ complete partition patterns and $10,777$ missing partition patterns. D_{Occ} is almost complete and generates $1,109$ complete partition patterns and 263 missing partition patterns for 611 tuples.

We designed a set of imputation rules over attributes in Tem and Occ. Rules have variable schemas, allowing to match with different query patterns. The set of rules in Table 4 is listed in priority order, the first rule is more pertinent (accurate) than the next one when both apply.

Some implicit (expert) knowledge about campus locations allowed us to define the priority order for some rules. Take the example of rules r_2 and r_3

Table 3. Data and pattern tables cardinalities

Variant x	Data D_x	Reference R_x	Complete P_x	Missing $\bar{P}_x$
Temp	1,321,686	24,615,600	11,268	10,777
Occ	10,131	10,742	1,109	263

Table 4. Imputation rules for sensor dataset *Temp*

building, floor, room:

rule	b	f	r	b	f	r	agg
r_0	3334	x_f	x_r	_	x_f	x_r	min(temp)
r_1	_	x_f	_	_	x_f	_	avg(temp)

building, floor, room, month, day:

rule	b	f	r	m	d	b	f	r	m	d	agg
r_2	x_b	x_f	10	x_m	x_d	x_b	x_f	12	x_m	x_d	temp
r_3	x_b	x_f	11	x_m	x_d	x_b	x_f	13	x_m	x_d	temp
r_4	x_b	x_f	x_r	8	_	x_b	x_f	x_r	_	_	max(temp)
r_5	x_b	x_f	x_r	x_m	_	x_b	x_f	x_r	x_m	_	avg(temp)
r_6	_	_	_	x_m	x_d	_	_	_	x_m	x_d	avg(temp)

building, floor, room, month, occupation:

rule	b	f	r	m	o	b	f	r	m	o	agg
r_7	x_b	x_f	x_r	_	"TD"	x_b	x_f	_	_	"TP"	avg(temp)
r_8	x_b	x_f	_	_	"TD"	x_b	x_f	_	_	_	avg(temp)

in Table 4: in the same floor, rooms are named sequentially in each side: one side with odd numbers, and the other with even numbers. Room 12 is then next room 10 (and not 11). The room planning allow rules such as r_9 since the usage of "TP" rooms is nearly the same (temporal scale) as for "TD", which explains the correlation between their temperature measures.

For our experiments, we define a set of aggregation queries over data tables *Temp* and $TempOcc = Temp \bowtie Occ$ (Table 5). All queries aggregate temperature measures along different attributes with variable filtering conditions (spatial, temporal, occupation, area).

Query Result Annotation: The query result annotation step consists in classifying each answer tuple as *correct*, *incorrect* and *missing*. We run an identification algorithm that implements functions *strict_match* and *weak_match* of Proposition 1 in Sect. 4. Table 6 classifies result patterns and partitions of each query in Table 5 into missing and correct categories. The answer data partitions are distributed between two classes *correct* and *incorrect*. Missing data is by definition not part of the query answer, since they do not belong to the data

Table 5. Analytical queries over sensor datasets

Q_1	**select** b, f, r, avg(temp) **from** Temp **where** b = 3334 **group by** b, f, r	Q_2	**select** b,f,r,m,d,**max**(temp) **from** Temp **where** m **in** (6,7,8) **group by** b, f, r, m, d
Q_3	**select** b,f,r,m,d avg(temp) **from** Temp **where** f **in** (4,5) **and** r **in** (10,11) **group by** b,m,d	Q_4	**select** b,f,r m, avg(temp) **from** TempOcc **where** b **in** (5354,5455) **and** o = "TD" **group by** b, f, r, m

Table 6. Correct, incorrect and missing patterns and data

	|Answer|	Correct		Incorrect		Missing		Time (sec)
		Patts	Data	Patts	Data	Patts	Data	
Q_1	8	0	0	8	8	24	108	1.6×10^{-2}
Q_2	1,012	119	1 012	0	0	132	256,588	10.0×10^{-2}
Q_3	1,602	4	377	7	1,225	116	5,333	2.9×10^{-2}
Q_4	44	19	22	22	22	66	220	4.3×10^{-2}

table (when using *null* values for representing missing information, missing data would correspond to *null* values in the result).

Observe that the number of patterns does not represent the number of corresponding data partitions. Pattern summarize completeness of data partitions at different sizes ([25,*,*] covers much more data than [25,1,10] which corresponds to one room partition). More generic patterns belong to a category set, wider they cover data partitions, and imputing this single patterns extends to all subsumed data. The running the running time is not impacted by the data table size (Q_3 vs. Q_4).

Query Result Imputation: The imputation strategy algorithm generates an ordered set of imputation queries to apply for each query "to repair" pattern set. Since the pattern summaries are shared by all imputation queries, we precompute the join between both data tables and the corresponding pattern tables and use the result in the imputation queries. Recall that rules are applied in the inverse order of their definition order. Take the example of the query Q_2. The ordered set of rules to repair the answer is $\{r_6, r_5, r_4, r_3, r_2\}$. The imputation process described in Sect. 4 is optimized in our experiments. Two imputation queries are executed. First, table *repairedPatt* stores an aggregation estimation obtained by joining the pattern table *torepair* with data table *available*. The obtained pattern table with freshly computed temperature values is then joined with the result table *Result* to generate the final table *repairedResult*. This pre-aggregation at the pattern level improves query performance since it avoids the redundant aggregation of partitions which are covered by the same patterns in the *Result*:

```
create table repairedPatt as
  select a.b, a.f, a.r, 8 as r.m, r.d, max(a.temp)
  from torepair r, available a
  where (a.b = r.b or r.b = '*') and (a.f = r.f or r.f = '*') and
          (a.r = r.r or r.r = '*')
  group by b, f, r, m, d

create table repairedResult as
  select r.b, r.f, r.r, p.m, r.d, p.temp
  from repairedPatt p, Result r
  where (r.b = p.b or p.b = '*') and (r.f = p.f or p.f = '*') and
          (r.r = p.r or p.r = '*') and (r.d=p.d or p.d = '*')
```

In Table 7, column *match patt.* records the number of patterns that can be repaired and column *cov. part.* shows the number of repaired partitions. The number of imputed partitions (column *imp. part.*) depends on the number of available correct partitions matching the rule's RHS for the repairing process. The number of remaining patterns (column *rem.*) corresponds to patterns that no rule has repaired.

Table 7. Imputation results

Query	rule	match. patt.	cov. part.	imp. part.	rem.	run time (10^{-3} sec)
Q_1	r_1	32	136	109	27	2.40
	r_0	32	136	40		1.58
Q_2	r_6	132	256,588	256588	0	27,910.00
	r_5	132	256,588	9936		720.00
	r_4	132	86459	10261		3,260.00
	r_3	25	9292	920		1.74
	r_2	25	10212	920		1.84
Q_3	r_6	127	6558	6558	0	13,890.00
	r_5	127	6558	1084		2,240.00
	r_4	25	465	10261		3.70
	r_3	123	5333	331		1,590.00
	r_2	74	1225	342		170.00
Q_4	r_8	88	242	242	0	4.78
	r_7	88	242	66		0.15

Observe from the set of rules that only r_1 and r_0 are applicable for the first query. We start by applying the rule r_1 with less priority, imputing 109 partitions over 136. The rule r_0 repairs less tuples, since it requires repairing a room with the average observed temperature for the same room during the year. Many rooms are not equipped with sensors at all which explains the poor number of imputation update achieved with this rule. At the end, 27 results still remain without any estimation. We found for example that all missing partitions matching the patterns $(3334, JU, *)$, $(3334, SS, *)$ and $(3334, SB, *)$ could not be imputed, since no temperature measure is available for these floors in all campus buildings. Note that both applied rules require a completion using the same floor, but no recording sensor is available for these floors, preventing imputation. All other queries could be repaired completely by applying all matching imputation rules. These experiments demonstrate that the usefulness of imputation rules depends on the existence of correct answers and the expert's knowledge about the sensor network configuration and behavior.

6 Conclusion

We presented a new query-driven imputation approach for repairing analytic query results using imputation rules. We propose a complete query rewriting process that starts from missing and incorrect data identification using completeness patterns to generate imputation strategies for estimating missing and incorrect query results. The current imputation model is limited to aggregation queries with equality predicates in disjunctive normal form and a first possible extension would be to extend matching to inequality predicates. A second extension concerns the introduction of statistical quality criteria like precision in rule selection process. One obvious criteria for choosing a rule might be the coverage of available correct data for estimating missing values. Finally, another research direction concerns the automatic generation of imputation rules by using data mining and machine learning techniques.

Acknowledgement. This work has partially been supported by the EBITA collaborative research project between the Fraunhofer Institute and Sorbonne Université.

References

1. Buck, S.F.: A method of estimation of missing values in multivariate data suitable for use with an electronic computer. J. R. Stat. Soc. Ser. B (Methodol) **22**, 302–306 (1960)
2. Cambronero, J., Feser, J.K., Smith, M.J., Madden, S.: Query optimization for dynamic imputation. Proc. VLDB Endowment **10**(11), 1310–1321 (2017)
3. Chu, X., Ilyas, I.F., Krishnan, S., Wang, J.: Data cleaning: overview and emerging challenges. In: Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data, pp. 2201–2206. ACM, New York (2016)

4. Chung, Y., Mortensen, M.L., Binnig, C., Kraska, T.: Estimating the impact of unknown unknowns on aggregate query results. ACM Trans. Database Syst. (TODS) **43**(1), 3 (2018)
5. Dallachiesa, M., et al.: NADEEF: a commodity data cleaning system. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 541–552. ACM (2013)
6. Fan, W.: Dependencies revisited for improving data quality. In: Proceedings of the 2008 ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 159–170. ACM (2008)
7. Fan, W., Geerts, F.: Relative information completeness. ACM Trans. Database Syst. (TODS) **35**(4), 27 (2010)
8. Fan, W., Li, J., Ma, S., Tang, N., Yu, W.: Towards certain fixes with editing rules and master data. Proc. VLDB Endowment **3**(1–2), 173–184 (2010)
9. Farhangfar, A., Kurgan, L., Dy, J.: Impact of imputation of missing values on classification error for discrete data. Pattern Recognit. **41**(12), 3692–3705 (2008)
10. Garofalakis, M.N., Gibbons, P.B.: Approximate query processing: taming the terabytes. In: Proceedings of 27th International Conference on Very Large Databases (VLDB), pp. 343–352 (2001)
11. Hannou, F.Z., Amann, B., Baazizi, A.M.: Exploring and comparing table fragments with fragment summaries. In: The Eleventh International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA). IARIA (2019)
12. Liao, Z., Lu, X., Yang, T., Wang, H.: Missing data imputation: a fuzzy k-means clustering algorithm over sliding window. In: 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery, vol. 3, pp. 133–137. IEEE (2009)
13. Mansinghka, V., Tibbetts, R., Baxter, J., Shafto, P., Eaves, B.: BayesDB: A probabilistic programming system for querying the probable implications of data. arXiv preprint arXiv:1512.05006 (2015)
14. Razniewski, S., Korn, F., Nutt, W., Srivastava, D.: 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, pp. 561–576, 31 May–4 June 2015
15. Silva-Ramírez, E.L., Pino-Mejías, R., López-Coello, M., Cubiles-de-la Vega, M.D.: Missing value imputation on missing completely at random data using multilayer perceptrons. Neural Netw. **24**(1), 121–129 (2011)
16. Wang, J., Krishnan, S., Franklin, M.J., Goldberg, K., Kraska, T., Milo, T.: A sample-and-clean framework for fast and accurate query processing on dirty data. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 469–480. ACM (2014)
17. Wang, J., Tang, N.: Towards dependable data repairing with fixing rules. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 457–468 (2014)
18. Zhu, B., He, C., Liatsis, P.: A robust missing value imputation method for noisy data. Appl. Intell. **36**(1), 61–74 (2012)