Consistent Aggregations in Databases with Referential Integrity Errors

Carlos Ordonez
University of Houston
Houston, TX, USA

Javier García-García
UNAM University
Mexico City, Mexico

ABSTRACT

A data warehouse integrates tables coming from multiple source databases, where each database has different tables, columns with similar content across databases and different referential integrity constraints, enforced to different compliance levels. Some source databases may have more reliable data than others, if referential integrity is more strictly enforced or their respective logical data model is more comprehensive. Thus, a query in an integrated database is likely to refer to tables and columns with referential integrity errors. In this work, we improve aggregations to handle referential integrity errors on OLAP databases. Specifically, when two tables are joined SQL ignores those tuples with invalid foreign key values, effectively discarding potentially valuable information. We extend aggregations to return complete answer sets in the sense that no tuple is excluded. Two families of extended aggregations are proposed: weighted referential aggregations and full referential aggregations, which return an approximate answer set and perform a dynamic repair, respectively. Finally, we introduce a simple method to improve aggregation accuracy. Experiments analyze approximation accuracy and time performance of our extended aggregations on a synthetic database, comparing them with standard SQL aggregations on databases with varying referential error rates. The extra work to compute extended aggregations is reasonable and approximate answer sets are highly accurate, making our aggregations a good alternative to standard aggregations in a data warehouse.

1. INTRODUCTION

There has been a growing interest on the problem of obtaining improved answer sets produced by queries in a setting where a database violates integrity constraints or has incomplete data. This is a common scenario in a data warehouse, where multiple databases of different reliability and similar contents are integrated. Database integration is by itself a deep problem [18]. A natural solution to the problem of getting consistent answer sets in incomplete and inconsistent databases is to repair the database either by removing inconsistent data or by fixing errors. The chief disadvantage about such approach is that the database must be modified. Removing data is the easiest, but generally not an acceptable, solution. On the other hand, fixing errors is difficult since it requires understanding inconsistencies across multiple tables, potentially going back to the source databases. An overview of data cleaning can be found in [8, 16]. In this work, we propose an alternative solution that does not modify the database. We introduce two families of extended aggregations that improve the answer set with additional information without discarding tuples with invalid foreign keys. The first family approximates consistent answer sets without updating the tables used in the query and the second one performs a dynamic repair on the answer set. Both families of aggregations are suitable for OLAP databases.

On a related topic, there have been two main themes on extending database models and defining new aggregate functions: getting accurate answers taking a long computation time or quickly returning approximate answers. A trade-off is always present when accuracy and speed compete with each other. To the best of our knowledge, this problem has not been studied in the presence of inconsistency and incompleteness on the foundation of the relational model: referential integrity.

This is an outline of the rest of our article. Section 2 presents definitions. Section 3 explains how to compute aggregations in the presence of referential integrity problems and introduces two families of extended aggregations. Section 4 presents experiments with synthetic databases. Section 5 discusses related research comparing it to our approach. Section 6 concludes the article.

2. DEFINITIONS

2.1 Relational Database

A relational database is denoted by $D(\mathcal{R}, I)$, where $\mathcal{R}$ is a set of N relations $\mathcal{R} = \{R_1, R_2, \ldots, R_N\}$, R_i is a set of tuples and I a set of referential integrity constraints. R_i is relation of degree d_i, where each attribute comes from some domain. We use simple primary keys (PKs) to make exposition simpler. In other words, primary keys consist of one attribute. One attribute is the primary key that we will also call K. The size of R_i (i.e. its number of tuples) is denoted by $|R_i|$ and n_i (to avoid confusion with N). Relations are manipulated with the standard relational algebra operators $\sigma, \Pi, \bowtie$ and aggregations, defined below.

2.2 Referential Integrity

A referential integrity constraint, belonging to I, between two relations R_i and R_j is a statement of the form: $R_i(K) \rightarrow R_j(K)$, where R_i is the referencing relation, R_j is the referenced relation, K is a *foreign key* (FK) in R_i and K is the primary key (PK) or a candidate key of R_j. In general, we

will refer to K as the primary key of R_j. To simplify exposition we assume the common attribute K has the same name on both relations R_i and R_j. In a valid database state with respect to I, the following two conditions hold for every referential constraint: (1) $R_i.K$ and $R_j.K$ have the same domains. (2) for every tuple in R_i there must exist a tuple in R_j such that $R_i.K = R_j.K$. The primary key of a relation ($R_j.K$ in this case) is not allowed to have nulls. But in general, for practical reasons the foreign key $R_i.K$ is allowed to have nulls when its value is not available at the time of insertion or when tuples from the referenced relation are deleted and foreign keys are nullified [9].

We relax the concept of referential integrity. We assume the database may be in an invalid state with respect to I. That is, some referential integrity constraints may be violated in subsets of $\mathcal{R}$. We refer to the valid state defined above as a *strict state*. A database state where there exist referential errors is called *relaxed state*. In a relaxed database R_i may contain tuples having $R_i.K$ values that do not exist in $R_j.K$ or $R_i.K$ may be null. To make our definition more precise, if $R_i.K$ in the referencing relation is null in some tuple we will consider such tuple incorrect. This is motivated by the fact that a null reference provides no information and the $\bowtie$ operator eliminates R_i tuples with a null value on K. We denote a generic null value by η.

2.3 Aggregations

Let ${}_B\mathcal{F}_{agg(A)}(R_i)$ be the answer set returned by an aggregation, where B is the grouping attribute, agg() is an aggregate function and A is some attribute to compute aggregations on. This notation can be easily generalized to more than one grouping attribute and two or more aggregate functions. There exist several different definitions for aggregate functions [14], which have distinct semantics for handling nulls, multi-sets (bags) and denormalization. From now on, unless stated otherwise, when we refer to an aggregate function agg(), it is taken from { count(), sum(), max(), min(), avg() } based on the standard SQL definition [11].

Assume we have an OLAP (multidimensional) database model in the form of a star schema for a data warehouse. This model has two kinds of tables: a fact table T and multiple surrounding dimension tables $D_1, \ldots, D_k$, referenced by foreign keys $K_1, \ldots, K_k$. We focus on star-joins to compute aggregations. That is, computing joins between the fact table and several dimension tables. Notice that the more general snow-flake join can be reduced to a star join and therefore we do not study it in our work. Going back to our relational database definitions, T is referenced as R_i and each dimension table is manipulated as R_j in a generic manner. The fact table has two kinds of attributes: dimension attributes (e.g. B), which reference the dimension tables and measure attributes (e.g. A), to be aggregated grouping by subsets of the dimension attributes. The following two relations will be used throughout the article:

$$R_i(\ldots, K, \ldots A, \ldots, B, \ldots), \qquad R_j(\underline{K}, \ldots, C, \ldots),$$

where R_i represents a referencing relation playing the role of the fact table and R_j represents a referenced relation acting as the dimension table. Attribute K is a foreign key in R_i and the primary key in R_j, A is a measure attribute and C is an attribute functionally dependent on $R_j.K$. We are particularly interested in computing aggregations of the form:

$${}_{R_j.C}\mathcal{F}_{agg(R_i.A)}\ (R_i \bowtie_K R_j)$$

or equivalently in SQL

```
SELECT R_j.C, agg(R_i.A)
FROM R_i JOIN R_j ON R_i.K = R_j.K
GROUP BY C
```

where agg() is an aggregate function, as defined above. The aggregation over attribute A associated to a specific value v of grouping attribute B will be denoted as

$${}_{B=v}\mathcal{F}_{agg(A)}(R_i) \tag{1}$$

or in an equivalent manner in SQL as

```
SELECT agg(R_i.A)
FROM R_i
GROUP BY B
HAVING R_i.B = v
```

To close this section, we define the value contained in the answer set of expressions like Equation 1, with one tuple and one attribute, as follows:

$$\mathcal{V}({}_{B=v}\mathcal{F}_{agg(A)}(R_i)).$$

In other words, $\mathcal{V}()$ is used to convert an atomic relation with one tuple and one attribute into one value to make it compatible for arithmetic and logical expressions. Notice this value will be null when agg($R_i.A$) is null (η).

2.4 Motivating Examples

Our examples throughout the article are based on a chain of stores database with three relations:
sales(storeId, cityId, regionId, salesamt,year, ...)
city(cityId, cityName, country, ...)
region(regionId, regionName, ...)
which are the result of integrating two databases from two companies, Cx and Rx, that are in a process of database integration. Cx had store information organized by city and Rx had it organized by region. Also, suppose that within a region there are several cities. Tables from both databases share a common key, *storeId* without conflicts. The integrated database in a relaxed state is shown in Figure 1, where invalid references are highlighted.

Example 1 *The attribute* cityId *in* sales *is a foreign key, but also it plays the role of a dimension attribute. The attribute* salesamt *is a measure attribute. The referential integrity constraint,* sales(cityId) $\rightarrow$ city(cityId)*, should hold between the two relations. Now observe the query in Figure 2. The unioned query computes the sales amounts grouped by* cityName *and the total sales amount. But as we can see from the second query in Figure 3 the answer set is inconsistent. That is, their total sum of sales amount is different:*

$$\mathcal{V}(\mathcal{F}_{sum(salesamt)}(sales)) \neq \mathcal{V}(\mathcal{F}_{sum(salesamt)}(sales \bowtie_{cityId} city))$$

In this example, consistency means a distributive function [10] preserves partial results from subsets of a set [14]. In this relaxed state the answer sets are inconsistent due to the existence of referential integrity errors. The first unioned query gives grouped aggregates, that considered as a whole,

sales

storeId	cityId	regionId	salesamt×1K	year	...
1	LAX	AM	540.30	2005	...
2	LAX	AM	640.25	2006	...
3	HOU	AM	434.36	2005	...
4	HOU	AM	440.80	2006	...
5	MON	AM	304.76	2006	...
6	MEX	AM	483.24	2006	...
7	**NXX**	AM	339.21	2005	...
8	**CXY**	AM	534.71	2005	...
9	η	AM	651.19	2006	...
10	SYD	AP	458.82	2005	...
11	SYD	AP	455.21	2006	...
12	MEL	AP	385.27	2006	...
13	ROM	EU	532.16	2005	...
14	ROM	EU	583.95	2006	...
15	PAR	EU	422.72	2005	...
16	PAR	EU	451.82	2006	...
17	MAD	EU	396.23	2006	...
18	FLR	EU	271.29	2006	...
19	HAM	EU	389.21	2005	...
20	HAM	EU	400.52	2006	...

city

cityId	cityName	country	...
FLR	Florence	IT	...
HAM	Hamburg	GE	...
HOU	Houston	US	...
LAX	Los Angeles	US	...
LON	London	UK	...
MAD	Madrid	SP	...
MEL	Melbourne	AU	...
MEX	Mexico	MX	...
MON	Montreal	CA	...
PAR	Paris	FR	...
ROM	Rome	IT	...
SYD	Sydney	AU	...

region

regionId	regionName	...
EU	Europe	...
AM	Americas	...
AP	Asia-Pacific	...

Figure 1: A store database in a relaxed state with invalid foreign keys highlighted.

are inconsistent with respect to the total given by the same query. The answer set represents the original database, but with the tuples with referential integrity errors deleted. Observe that invalid tuples are eliminated from the aggregate function answer set.

Example 2 *Notice that based on the data warehouse integrity constraints, the functional dependency:* sales.cityId → sales.regionId*, should hold. Suppose the information from Rx is more reliable than information from Cx.*

A query getting total sales by region and total overall sales is shown in Figure 4. In this case, a consistent answer set is obtained joining with the foreign key regionId*, in contrast to the inconsistency mentioned above in Example 1. If we know the functional dependency* sales.cityId → sales.regionId *holds, could we obtain an improved answer set when grouping by attribute* sales.cityId *in the presence of invalid foreign key values? This will be the goal of our approach.*

3. AGGREGATIONS IN DATABASES WITH REFERENTIAL INTEGRITY ERRORS

We present definitions for referential frequency, referential weight and weighted referential partial probability. Based on those definitions we introduce two families of SQL extended aggregations: weighted referential aggregations and full referential aggrations.

3.1 Extended Aggregations

We start by defining *referential frequency*, *referential weight*, *referential partial probability* and *weighted referential partial probability*. For the following definitions, consider a relaxed database where the referential integrity constraint $R_i(K) \rightarrow R_j(K)$ could be violated.

The *referential frequency* of a correct referencing value in a given foreign key refers to the number of tuples in the

```
SELECT cityName, sum(salesamt)
 FROM sales
        JOIN city ON sales.cityId=city.cityId
 GROUP BY cityName
UNION
SELECT '-TOTAL', sum(salesamt)
 FROM sales
```

cityName	*sum*()
Florence	271.29
Hamburg	789.73
Houston	875.16
Los Angeles	1180.55
Madrid	396.23
Melbourne	385.27
Mexico	483.24
Montreal	304.76
Paris	874.54
Rome	1116.11
Sydney	914.03
-TOTAL	9116.02

Figure 2: Inconsistent answer set (total is incorrect).

```
SELECT sum(salesamt)
FROM sales JOIN city
ON sales.cityId=city.cityId
```

sum()
7590.91

Figure 3: Total sales from valid references.

```
SELECT regionName, sum(salesamt)
FROM sales
    JOIN region ON sales.regionId=region.regionId
GROUP BY regionName
UNION
SELECT '-TOTAL', sum(salesamt)
FROM sales
```

regionName	*sum()*
Europe	3447.90
Americas	4368.82
Asia-Pacific	1299.30
-TOTAL	9116.02

Figure 4: Consistent answer set.

referencing relation that hold this correct value. The *referential frequency* of a given foreign key refers to the number of tuples with correct references in the foreign key.

Definition 1 *(Referential frequency) We define the aggregate function* referential frequency *over a foreign key K referencing $R_j.K$, $\mathcal{F}_{reff_{R_j.K(K)}}(R_i)$ where subscript $R_j.K$ is the referenced key, as*

$$\mathcal{F}_{reff_{R_j.K(K)}}(R_i) = \mathcal{F}_{count(K)}(R_i \bowtie_K R_j)$$

Let $k_c \in \Pi_K(R_i \bowtie_K R_j)$ be a correct reference. Using the notation defined in Section 2.3 to reference one element of the aggregate list, we compute $\mathcal{V}(_{K=k_c}\mathcal{F}_{reff_{R_j.K(K)}}(R_i))$ as follows

$$\begin{aligned}\mathcal{V}(_{K=k_c}\mathcal{F}_{reff_{R_j.K(K)}}(R_i)) &= \mathcal{V}(_{K=k_c}\mathcal{F}_{count(K)}(R_i \bowtie_K R_j) \\ &= |\sigma_{K=k_c}(R_i \bowtie_K R_j)|\end{aligned}$$

The next equation links the value of the referential frequency of a given foreign key and its values.

$$\begin{aligned}&\mathcal{V}(\mathcal{F}_{reff_{R_j.K(K)}}(R_i)) \\ &\quad = \Sigma_{k_c \in \Pi_K(R_i \bowtie_K R_j)}(\mathcal{V}(_{K=k_c}\mathcal{F}_{reff_{R_j.K(K)}}(R_i))).\end{aligned}$$

Observe in a relaxed database, when $R_i.K$ only has invalid references, or $R_i = \emptyset$ then $\mathcal{V}(\mathcal{F}_{reff_{R_j.K(K)}}(R_i)) = 0$.

Example 3 *Consider the database in relaxed state of Figure 1. For value* SYD *we have*

$$\mathcal{V}(_{cityId=SYD}\mathcal{F}_{reff_{city.cityId(cityId)}}(sales)) = 2,$$

which equals the number of tuples that have SYD, a valid foreign key value. For the foreign key sales.cityId *we have*

$$\mathcal{V}(\mathcal{F}_{reff_{city.cityId(cityId)}}(sales)) = 17,$$

which is the total number of tuples that hold correct references in the foreign key sales.cityId.

The *referential weight* of a correct reference of a given foreign key refers to the fraction of tuples that hold this correct reference over the total number of tuples with correct references. That is, we define referential weight in terms of referential frequency as follows:

Definition 2 *(Referential weight) We define the aggregate function* referential weight *over a foreign key K as*
$\mathcal{F}_{refw_{R_j.K(K)}}(R_i)$ where for each correct reference
$k_c \in \Pi_K(R_i \bowtie_K R_j)$ the aggregate value is computed as follows

$$\mathcal{V}(_{K=k_c}\mathcal{F}_{refw_{R_j.K(K)}}(R_i)) = \frac{\mathcal{V}(_{K=k_c}\mathcal{F}_{reff_{R_j.K(K)}}(R_i))}{\mathcal{V}(\mathcal{F}_{reff_{R_j.K(K)}}(R_i))}$$

The domain for the values of the elements of the referential weight *is the rational numbers in [0,1]. We compute $\mathcal{V}(\mathcal{F}_{refw_{R_j.K(K)}}(R_i))$ as follows*

$$\begin{aligned}&\mathcal{V}(\mathcal{F}_{refw_{R_j.K(K)}}(R_i)) \\ &\quad = \Sigma_{k_c \in \Pi_K(R_i \bowtie_K R_j)}(\mathcal{V}(_{K=k_c}\mathcal{F}_{refw_{R_j.K(K)}}(R_i))) \\ &\quad = 1\end{aligned}$$

if there is at least one valid reference in $R_i.K$. When the values of $R_i.K$ are all invalid references, or R_i is empty then $\mathcal{V}(\mathcal{F}_{refw_{R_j.K(K)}}(R_i)) = 0$.

The next definition is in the spirit of the definition of partial probability in [15]. The idea behind the *referential partial probability* is to associate to each correct (referencing) value in a given foreign key and to each unreferenced value in the referenced key a probability. Each probability corresponds to the probability that the associated value be the correct reference in a tuple with an invalid value in the foreign key. In [15] the authors define a partial probability as a vector of probabilities. Each probability is associated to a value. Each probability corresponds to a probability that an attribute value be the corresponding value associated to the probability in the partial probability vector. Formally we have the following definition.

Definition 3 *(Referential partial probability) The* referential partial probability $\text{refpp}_{R_j.K}(R_i.K)$ *defined over the correct referencing values of a foreign key $R_i.K$ and over the unreferenced values in $R_j.K$ with respect to $R_i.K$, is a vector of probabilities where, if there are correct references in $R_i.K$, we have $\langle p_1, p_2, \ldots, p_s, \ldots, p_t \rangle$,*
$s = |\Pi_K(R_i \bowtie_K R_j)|$, $t = |R_j|$, where each p_r is the probability that an invalid reference be the $k_r, r = 1, \ldots, s$ correct reference in $R_i.K$ or the $r = s+1 \ldots t$ unreferenced value in $R_j.K$ For the cases where all references in $R_i.K$ are invalid, or R_i is empty, that is $s = 0$, the vector will only hold the probabilities that correspond to the (unreferenced) values of $R_j.K$. If $|R_j| = 0$, then the referential partial probability *is undefined. The element of the referential partial probability vector that corresponds to a k_r value will be denoted as* $\text{refpp}_{R_j.K}(R_i.K, k_r)$.

Notice that for each referenced key, a set of partial probability vectors may be defined, one for each foreign key that references it. Users with good database knowledge may assign these probabilities. Nevertheless, a feasible way to assign these probabilities when computing our proposed aggregate functions is following the intuition that a high probability will correspond to a high frequency in the foreign key

and a low probability corresponds to a low frequency. With these ideas in mind, we can see that the *referential weight* may be used to define a *referential partial probability* vector.

Definition 4 *(Weighted referential partial probability) We define the* weighted referential partial probability *of a foreign key $R_i.K$ referencing $R_j.K$,* $\text{wrpp}_{R_j.K}(R_i.K)$*, as the referential partial probability where each one of its elements,* $\text{wrpp}_{R_j.K}(R_i.K, k_r)$ *is computed as follows:*

Case 1. There is at least one valid reference in $R_i.K$
$\text{wrpp}_{R_j.K}(R_i.K, k_r)$

$$= \begin{cases} \mathcal{V}(_{K=k_r}\mathcal{F}_{refw_{R_j.K}(K)}(R_i)), \\ \qquad if\ k_r \in \Pi_K(R_i \bowtie_K R_j) \\ \\ 0 \qquad if\ k_r \in \Pi_K(R_j) - \Pi_K(R_i \bowtie_K R_j) \end{cases}$$

Case 2. The values in $R_i.K$ are all invalid references or $R_i.K$ is empty

$$\text{wrpp}_{R_j.K}(R_i.K, k_r) = \frac{1}{|R_j|}, k_r \in \Pi_K(R_j)$$

Observe that we are assuming a uniform probability distribution function in case 2, since we do not have more information. Thus defined then $wrpp_{R_j.K}(R_i.K)$ has the following property: $\Sigma_{k \in R_j.K}(wrpp_{R_j.K}(R_i.K, k)) = 1$.

Example 4 *Consider the relaxed database of Figure 1. The* weighted referential partial probability *defined over the correct references of foreign key* sales.cityId *and over the unreferenced values in* city.cityId *with respect to* sales.cityId*,* ⟨LAX,HOU,MON,MEX,SYD,MEL,ROM,PAR,MAD, FLR,HAM,LON⟩ *is:*

$$wrpp_{city.cityId}(sales.cityId) = \langle \frac{2}{17}, \frac{2}{17}, \frac{1}{17}, \frac{1}{17}, \frac{2}{17}, \frac{1}{17}, \frac{2}{17}, \frac{2}{17}, \frac{1}{17}, \frac{1}{17}, \frac{2}{17}, 0 \rangle$$

To simplify exposition and notation, from now on when there exists at least one valid reference in the referencing foreign key we will show only those values different from 0 from the *weighted referential partial probability* vector. We will not refer to each value correspondence when it is obvious from the context. Next, we propose two families of new aggregate functions:

- weighted referential aggregations
- full referential aggregations

The general idea of our proposed *weighted referential aggregations* is the following. First, each invalid reference will be treated as an imprecise value. That is, a value we know it belongs to a given set of values, in this case the set of referenced values at the time the aggregate function is evaluated. The values of the measure attribute of the invalid references account for part of the corresponding value of each valid reference, according to their probability in the *weighted referential partial probability* vector, in particular, according to their *referential weight* value. We introduce a method to improve the answer sets returned by the extended aggregate functions by means of a reliable attribute.

The *full referential aggregations* are the counterpart of the SQL grouped attribute aggregations computed over a joined relation on foreign key-primary key attributes, from now on SQL joined aggregations, with possible referential integrity violations. All the values of the measure attribute of the invalid references account for part of the corresponding value of each group the way it will be explained. Observe that *full referential aggregations* may not be consistent with respect to the total aggregate value, just like standard SQL joined aggregations are not consistent either when computed over a database in a relaxed state, as shown in Example 1. Nevertheless, these values represent for each group a possible correct aggregate value. That is, they represent a possible world for each group as described in a different context in [3] and [1]. The set of values in Example 1 correspond to the case where repairing the database resulted in the elimination of all the tuples with an invalid reference and none of them ended being part of a group. The answer sets of *full referential aggregations* correspond to the case where for each group, all invalid references were repaired and updated with the corresponding reference of the group.

cityName	sum()	wr_sum()	fr_sum()
Florence	271.29	361.00	1796.40
Hamburg	789.73	969.15	2314.84
Houston	875.16	1054.58	2400.27
Los Angeles	1180.55	1359.97	2705.66
Madrid	396.23	485.94	1921.34
Melbourne	385.27	474.98	1910.38
Mexico	483.24	572.95	2008.35
Montreal	304.76	394.47	1829.87
Paris	874.54	1053.96	2399.65
Rome	1116.11	1295.53	2641.22
Sydney	914.03	1093.45	2439.14

Table 1: Weighted referential and full referential sum aggregations.

Example 5 *Consider the aggregate function* sum() *over the measure attribute* salesamt *grouped by* cityName *as in Figure 2. The answer set of the* weighted referential sum aggregation*, denoted as* wr_sum()*, is shown in Table 1. The user has a consistent answer set, where the value of the measure attribute of the tuples with invalid references is distributed among the correct references according to the weighted referential partial probability vector. On the other hand, the* full referential sum aggregation*, denoted as* fr_sum() *in the same Table, gives an upper bound of possible sales amounts for each group considering the invalid references, but it is not a consistent answer set.*

3.2 Weighted Referential Aggregations

We consider weighted referential aggregate functions that correspond to the following aggregations: count(*), count() and sum(). They will be denoted with the prefix wr and the same name as their SQL counterparts. Due to the particular properties of aggregate functions min(), max() and avg(), only in a few cases the weighted referential versions are meaningful. Therefore, we omit their definition.

We will base our definitions in the *weighted referential partial probability*, particularly the case where at least there is one valid reference, but these definitions may be generalized to consider other *referential partial probability vectors*.

3.2.1 Function Properties

Our extended aggregate functions must fulfill certain properties to be considered clean extensions of their counterpart standard SQL aggregations. An *ascending* feature as defined in [12] holds for the *weighted referential aggregations* wr_count(*) and wr_count(). In [12], the authors define an ascending aggregate function f as a function where if given sets of records S and $S^{'}$, being $S^{'} \subseteq S$ then $f(S^{'}) \leq f(S)$. Descending functions are defined accordingly. That is, in our context, as tuples are inserted or deleted, the *referentially weighted aggregations* may increase (i.e. ascending) or decrease (i.e. descending). For wr_sum() aggregate function, there are cases where inserting or deleting tuples implies an increasing or decreasing aggregate as in many OLAP scenarios (Example 1). In these cases the aggregate function wr_sum() fulfills an *ascending* or *descending* feature. Observe that as the referential integrity errors are repaired with correct references without varying the number of tuples, we will expect that the element of the aggregate function list of each group remains more or less constant. We will also expect that the total value remains constant in this type of repairing processes.

If the referential integrity errors are repaired or if there are no referential errors. That is, if the integrity constraint holds for all tuples, a $safety$ feature holds for the *weighted referential aggregations*, meaning that the answer sets will not be different compared to the the ones from the standard SQL aggregations.

For the distributive functions, wr_count(*), wr_count() and wr_sum(), a *consistency* property holds if there is at least one correct reference, as we will see next. This property corresponds to the *summarizability* feature described in [13]. That is, a distributive function over a set should preserve the results over the subsets of its partitions.

Finally, since we are looking for feasible computations the *weighted referential aggregations* will return the same answer set as the standard SQL aggregations when there are no valid references. Such answer set represents the ground fact that we start with an empty relation or with a totally erroneous foreign key. Notice we could have defined the *weighted referential aggregations* considering the second case in Definition 4. Although consistency would have been satisfied even for the facts described above, efficiency would have been compromised since all the values in the referenced attribute would appear in the answer set with a value that corresponds to an equal probability. When in the definition of the *weighted referential aggregation* the second case of Definition 4 is considered, we will refer to these variants as the *completely consistent weighted referential aggregations.*

3.2.2 Function Definitions

We will now define the *weighted referential aggregations* considering the case there is at least one valid reference. We start defining the *weighted referential count(*)* denoted as wr_count(*). First we define $\mathcal{F}_{wr\_count_{R_j.K(*)}}(R_i)$, as

$$\mathcal{F}_{wr\_count_{R_j.K(*)}}(R_i) = \mathcal{F}_{count(*)}(R_i)$$

Now we show how to compute each element of the aggregate function list. For each correct reference k_c of foreign key $R_i.K$ that references $R_j.K$, the aggregate value is computed as follows:

$$\begin{aligned}
&\mathcal{V}(_{K=k_c}\mathcal{F}_{wr_count_{R_j.K(*)}}(R_i)) \\
&\quad = \mathcal{V}(_{K=k_c}\mathcal{F}_{reff_{R_j.K(*)}}(R_i)) \\
&\qquad + (\mathcal{V}(\mathcal{F}_{count(*)}(R_i)) - \mathcal{V}(\mathcal{F}_{reff_{R_j.K(*)}}(R_i))) \\
&\qquad * \mathcal{V}(_{K=k_c}\mathcal{F}_{refw_{R_j.K(K)}}(R_i))
\end{aligned}$$

See how the number of invalid references,
$\mathcal{V}(\mathcal{F}_{count(*)}(R_i)) - \mathcal{V}(\mathcal{F}_{reff_{R_j.K(*)}}(R_i))$,
account for part of the corresponding aggregate of the valid references via the *referential weight* value -
$\mathcal{V}(_{K=k_c}\mathcal{F}_{refw_{R_j.K(K)}}(R_i))$. Observe that the elements of the aggregate list are rational numbers in $[1, \infty)$. Thus defined wr_count(*) is *consistent* if at least there is one correct reference since by Definitions 1 and 2 we have

$$\begin{aligned}
&\Sigma_{k_c \in R_i.K}(|\sigma_{_{K=k_c}}(R_i \bowtie_K R_j)| \\
&\quad + (|R_i| - |R_i \bowtie_K R_j|) * \tfrac{|\sigma_{K=k_c}(R_i \bowtie_K R_j)|}{|R_i \bowtie_K R_j|})) \\
&= \Sigma_{k_c \in R_i.K}(|\sigma_{_{K=k_c}}(R_i \bowtie_K R_j)|) \\
&\quad + (|R_i| - |R_i \bowtie_K R_j|) * \Sigma_{k_c \in R_i.K}(\tfrac{|\sigma_{K=k_c}(R_i \bowtie_K R_j)|}{|R_i \bowtie_K R_j|})) \\
&= |R_i \bowtie_K R_j| + (|R_i| - |R_i \bowtie_K R_j|) \\
&= |R_i|
\end{aligned}$$

wr_count(*) has the *ascending* property mentioned above since its value increases as tuples are inserted. Also observe that as the invalid references, in this case $|R_i| - |R_i \bowtie_K R_j|$, decrease the total value remains constant. We can see it also fulfills the *safety* property. That is, if there are no referential integrity errors, the answer sets of both the SQL aggregate function count(*) and the *weighted referential count(*)* are the same.

In a similar fashion we define the *weighted referential count()*, wr_count() as

$$\mathcal{F}_{wr\_count_{R_i.K(K)}}(R_i) = \mathcal{F}_{count(K)}(R_i),$$

whose aggregate value is computed as follows:

$$\begin{aligned}
&\mathcal{V}(_{K=k_c}\mathcal{F}_{wr_count_{R_j.K(K)}}(R_i)) \\
&\quad = \mathcal{V}(_{K=k_c}\mathcal{F}_{reff_{R_j.K(K)}}(R_i)) \\
&\qquad + (\mathcal{V}(\mathcal{F}_{count(K)}(R_i)) - \mathcal{V}(\mathcal{F}_{reff_{R_j.K(*)}}(R_i))) \\
&\qquad * \mathcal{V}(_{K=k_c}\mathcal{F}_{refw_{R_j.K(K)}}(R_i))
\end{aligned}$$

The aggregation wr_count() has the *ascending*, *consistent* and *safety* properties. Likewise, we define wr_sum() as

$$\mathcal{F}_{wr\_sum_{R_i.K(A)}}(R_i) = \mathcal{F}_{sum(A)}(R_i),$$

whose aggregate value is computed as follows:

$$\begin{aligned}
&\mathcal{V}(_{K=k_c}\mathcal{F}_{wr_sum_{R_j.K(A)}}(R_i)) \\
&\quad = \mathcal{V}(_{K=k_c}\mathcal{F}_{sum(A)}(R_i)) \\
&\qquad + (\mathcal{V}(\mathcal{F}_{sum(A)}(R_i)) \\
&\qquad\quad - \Sigma_{k_r \in \Pi_K(R_i \bowtie_K R_j)}(\mathcal{V}(_{K=k_r}\mathcal{F}_{sum(A)}(R_i))) \\
&\qquad * \mathcal{V}(_{K=k_c}\mathcal{F}_{refw_{R_j.K(K)}}(R_i)),
\end{aligned}$$

where the addition and subtraction operators behave like the SQL sum() aggregation in the presence of η and the multiplication operator returns η, when some operand is η.

```
/* SQL query calling extended aggregation */
SELECT cityName, sum(salesamt), wr_sum(salesamt)
FROM city JOIN sales
           ON sales.cityId = city.cityId
GROUP BY cityName;

/* SQL statements evaluating extended aggregation */
CREATE TABLE wr_sumtemp AS
SELECT cityName, city.cityId AS fk,
  count(*) AS cardinality,
  count(sales.cityId) AS freq,
  sum(sales.salesamt) AS sumagg
FROM city RIGHT OUTER JOIN sales
  ON sales.cityId = city.cityId
GROUP BY cityName, city.cityId;

SELECT cityName,sum(sumagg), sum(wrsum)
FROM
(SELECT cityName
       ,((sumagg+(freq/
         (SELECT sum(freq) FROM wr_sumtemp
          WHERE fk IS NOT NULL ))*
(CASE WHEN
 (SELECT sumagg FROM wr_sumtemp WHERE fk is null)
   is null
 THEN 0 ELSE
 (SELECT sumagg FROM wr_sumtemp WHERE fk is null)
 END ))
  AS wrsum, sumagg
FROM wr_sumtemp
WHERE wr_sumtemp.fk is not null) AS foo
GROUP BY cityName;
```

Figure 5: Query calling wr_sum() and SQL statements evaluating the extended aggregation.

As stated above, as it is common in an OLAP scenario, wr_sum() makes sense when inserting or deleting tuples in R_i causes the value of the corresponding elements of the aggregate list to increase or decrease; wr_sum() fulfills the *consistent* and *safety* property. In Figure 5 we show a query in SQL calling wr_sum() and the equivalent SQL expressions obtaining the same answer set, assuming there exists at least one valid reference in foreign key $cityId$.

3.3 Method to Improve Weighted Referential Aggregations

We can improve and refine the estimated answer sets of the *weighted referential aggregations* if we have another foreign key or another attribute with values of higher quality in the same relation (i.e. an attribute with zero or less referential errors). This scenario is possible when two or more databases are integrated and there are relations that share a common primary key. A functional dependency must be defined between the two attributes and the dependency may be in either direction. Although the database is not in 3NF, remember we are supposing a relaxed database and our goal now is to keep all data, instead of repairing it. Suppose we have two foreign keys $R_i.K_a$ referencing $R_{j_a}.K_a$ and $R_i.K_b$ referencing $R_{j_b}.K_b$ and a measure attribute $R_i.A$ in a relaxed database. Also suppose the following functional dependency should hold between both attributes: $R_i.K_a \rightarrow R_i.K_b$. We can imagine this situation as if a set of elements represented by values in attribute $R_i.K_a$, e.g. cities, should be contained in an element represented by a value of $R_i.K_b$, e.g. a region.

First, consider the case where the user knows that the data quality of foreign key $R_i.K_b$ is higher than the quality of $R_i.K_a$. As before, the invalid references of foreign key $R_i.K_a$ are considered as imprecise values, but now we know these values represent elements that should be contained in an element represented by a value, say k_b, of foreign key $R_i.K_b$. That is, a subset of the correct references of $R_i.K_a$, more precisely the following values

$$\{k | k \in \Pi_{K_a}(\sigma_{K_b=k_b}(R_i \bowtie_{K_a} R_{j_a})\}$$

This fact reduces the set of values that the imprecise reference could stand for. Foreign key $R_i.K_b$ defines a partition of the values of $R_i.K_a$.

Example 6 *Look at the example of the relaxed database in Figure 1. Observe the tuple with value* 9 *in* sales.storeId. *As discussed before, the* weighted referential partial probability vector *may be used to compute how much a value that corresponds to an invalid reference accounts for in the corresponding value of each valid reference. In this case, the measure attribute* sales.salesamt *that corresponds to the invalid reference* η *(null) in* sales.cityId *should participate in each valid reference according to the weighted referential partial probability vector. So far, this is our best estimate. Now, since the user trusts the foreign key* sales.regionId*, the invalid reference mentioned above has a high probability that its 'real' value be a city in the* Americas *region.So the value of the measure attribute* sales.salesamt 651.19 *should participate only in each valid reference of the* Americas *region.*

Let us analyze the case when the user trusts the foreign key $R_i.K_a$. To fix a correct reference value instead of an invalid reference in $R_i.K_b$, we only need to know the dependent value, as specified by the functional dependency.

CASE	$\mathbf{R_i.K_a}$	$\mathbf{R_i.K_b}$	$\mathbf{R_i.A}$ values or tuples grouped by
1	$k_a \in \Pi_{R_i.K_a}(R_i \bowtie_{K_a} R_{j_a})$ k_a is valid	$k_b \in \Pi_{R_i.K_b}(R_i \bowtie_{K_b} R_{j_b})$ k_b is valid	k_a
2	$k_a \in \Pi_{R_i.K_a}(\sigma_{R_i.K_b=k_b}(R_i)$ $-\Pi_{R_i.*}(R_i \bowtie_{K_a} R_{j_a}))$ $\exists k \in \Pi_{R_i.K_a}(\sigma_{R_i.K_b=k_b}(R_i \bowtie_{K_a} R_{j_a}))$ k_a is invalid $\exists$ valid ref. in subset by k_b	$k_b \in \Pi_{R_i.K_b}(R_i \bowtie_{K_b} R_{j_b})$ k_b is valid.	$k \in \Pi_{R_i.K_a}(\sigma_{R_i.K_b=k_b}(R_i \bowtie_{K_a} R_{j_a}))$
3	$k_a \in \Pi_{R_i.K_a}(\sigma_{R_i.K_b=k_b}(R_i)$ $-\Pi_{R_i.*}(R_i \bowtie_{K_a} R_{j_a}))$ $\nexists k \in \Pi_{R_i.K_a}(\sigma_{R_i.K_b=k_b}(R_i \bowtie_{K_a} R_{j_a}))$ k_a is invalid $\nexists$ valid ref. in subset by k_b	$k_b \in \Pi_{R_i.K_b}(R_i \bowtie_{K_b} R_{j_b})$ k_b is valid.	$k \in \Pi_{R_i.K_a}(R_i \bowtie_{K_a} R_{j_a})$
4	$k_a \in \Pi_{R_i.K_a}(R_i \bowtie_{K_a} R_{j_a})$ k_a is valid	$k_b \in \Pi_{R_i.K_b}(R_i$ $-\Pi_{R_i.*}(R_i \bowtie_{K_b} R_{j_b}))$ k_b is invalid	k_a
5	$k_a \in \Pi_{R_i.K_a}(\sigma_{R_i.K_b=k_b}(R_i)$ $-\Pi_{R_i.*}(R_i \bowtie_{K_a} R_{j_a}))$ k_a is invalid	$k_b \in \Pi_{R_i.K_b}(R_i$ $-\Pi_{R_i.*}(R_i \bowtie_{K_b} R_{j_b}))$ k_b is invalid	$k \in \Pi_{R_i.K_a}(R_i \bowtie_{K_a} R_{j_a})$

Table 2: Cases for foreign key to improve the weighted referential aggregations - $R_i.K_a \rightarrow R_i.K_b$**.**

In both cases we have to consider a relaxed database. That is, we expect referential errors even in the 'trusted' foreign key. Also, the functional dependency constraint may be violated. A feasible approach towards getting better answer sets in the line of the aggregate functions proposed so far is the following.

Consider foreign keys $R_i.K_a$ and $R_i.K_b$ and the measure attribute $R_i.A$ and the functional dependency $R_i.K_a \rightarrow R_i.K_b$. We divide our exposition in two parts. First, we assume we know the value correspondence in the functional dependency. From the pairs of values that define the functional dependency we can derive a partition of a set of correct references of $R_i.K_a$. The set of valid references in $R_i.K_b$ defines a partition of the corresponding set of correct references in $R_i.K_a$. Next, a set of tuples in R_i may be associated to each correct reference, say k_b in $R_i.K_b$. Observe that these tuples may have invalid references in foreign key $R_i.K_a$. The values of attribute $R_i.A$ that correspond to these invalid tuples or the number of these tuples, in case we are dealing with the count() aggregations, will participate in each valid reference of the group of values in $R_i.K_a$ defined by k_b in $R_i.K_b$. This can be done computing the *weighted referential partial probability vector* considering $\sigma_{R_i.K_b=k_b}(R_i)$ as the referencing relation.

More precisely, suppose that for a given valid reference k_b in $R_i.K_b$, there exists a set of valid references, at least one, in $R_i.K_a$ that is determining the former one. For each valid reference k_a in $R_i.K_a$ that corresponds to this valid reference k_b in $R_i.K_b$ we have

$$wrpp_{R_{j_a}.K_a}(\sigma_{R_i.K_b=k_b}(R_i).K_a, k_a) = \frac{\mathcal{V}(_{K=k_a}\mathcal{F}_{reff_{R_{j_a}.K}(K)}(R_i))}{\Sigma_{k_s \in \Pi_{R_i.K_a}(\sigma_{R_i.K_b=k_b}(R_i))}(\mathcal{V}(_{K=k_s}\mathcal{F}_{reff_{R_{j_b}.K}(K)}(R_i)))}$$

Since $k_a \in \Pi_{R_i.K_a}(\sigma_{R_i.K_b=k_b}(R_i))$ is a valid reference the divisor expression does not evaluate to 0.

Example 7 *Again, take for instance the relaxed database in Figure 1. Due to the functional dependency* cityId → regionId*, the valid references of foreign key* regionId *(*$\{AM, AP, EU\}$*) define the following partition of valid references for foreign key* cityId *in table* sales*:*

$$\{\{LAX, HOU, MON, MEX\}, \{SYD, MEL\}, \{ROM, PAR, MAD, FLR, HAM\}\}$$

We can define three weighted referential partial probability vectors*, considering the three relations, each one of them with the tuples that have the values of each of the above subsets in foreign key* cityId*. Taking relation* sales *in Figure 1 the corresponding vectors are:*

$$\langle\frac{2}{6}, \frac{2}{6}, \frac{1}{6}, \frac{1}{6}\rangle \quad \langle\frac{2}{3}, \frac{1}{3}\rangle \quad \langle\frac{2}{8}, \frac{2}{8}, \frac{1}{8}, \frac{1}{8}, \frac{2}{8}\rangle$$

Contrast these three vectors with the vector shown in Example 4.

Table 2 shows the cases that should be considered depending on the values of $R_i.K_a$ and $R_i.K_b$ in each tuple and how should A value or tuple participate in an aggregate function in order to improve the estimated aggregate answer sets preserving the *consistency*, *ascending/descending* and *safety* properties in the aggregate functions where these properties apply. In Table 2 $R_i.*$ represents the list of all attributes in R_i.

So far we have assumed we know the correspondence between the values of $R_i.K_a$ and $R_i.K_b$ according to the functional dependency $R_i.K_a \rightarrow R_i.K_b$. But this is not a realistic assumption since we are dealing with a relaxed database. Assume there are violations to the functional dependency. In order to reconstruct feasibly the functional dependency so we can apply the strategy explained above, we can follow the intuition that a dependency violation appears with a much less frequency than a correct functional dependency. On the other hand, a pair of values of $R_i.K_a$ and $R_i.K_b$ that appear frequently associated in a number of tuples may be considered as a correct pair of values according to the func-

cityName	sum()	wr_sum()	wr_sum() improved
Florence	271.29	361.00	271.29
Hamburg	789.73	969.15	789.73
Houston	875.16	1054.58	1383.53
Los Angeles	1180.55	1359.97	1688.92
Madrid	396.23	485.94	396.23
Melbourne	385.27	474.98	385.27
Mexico	483.24	572.95	737.43
Montreal	304.76	394.47	558.95
Paris	874.54	1053.96	874.54
Rome	1116.11	1295.53	1116.11
Sydney	914.03	1093.45	914.03

Table 3: Weighted referential aggregate sum improved with trusted foreign key $regionId$.

tional dependency constraint. With these ideas in mind, we can reconstruct the functional dependency by choosing for each correct reference k_a in $R_i.K_a$ the correct reference k_b in $R_i.K_b$ to which k_a is associated the most. Ties are solved simply choosing one value. According to Table 2, if there is not a correct reference k_b in $R_i.K_b$ for k_a, then the tuples with a k_a reference in $R_i.K_a$ belong to case 4. If there are tuples with a k_a in $R_i.K_a$ associated to a correct reference k_b in $R_i.K_b$ but this pair of values was not the maximum pair of values for value k_a then these tuples will be treated as belonging to cases 2 or 3 since the user trusts foreign key $R_i.K_b$ so we assume k_a is an error. We show in Table 3 how the estimated aggregations given in Figure 1 may be improved by means of foreign key $regionId$.

Now, if the trusted foreign key is $R_i.K_a$, we proceed in a similar fashion. A correct reference of foreign key K_a determines only one value of K_b. If a pair of correct values k_a, k_b have not the maximum frequency, reference in attribute K_b will be considered an invalid value.

3.4 Full Referential Aggregations

The answer sets of our *full referential aggregations* represent for each element in the aggregate list, a potential repair which consists in updating each invalid reference with the value of the correct reference that represents each group. The answer sets from a standard SQL joined aggregation represent a potential repair of the database for each group that consists in deleting all the tuples with invalid references.

We will consider *full referential aggregations* that correspond to the aggregate functions count(*), count(), sum(), max(), min(), and avg() and will be denoted as their counterparts but with the prefix fr. The properties that the *full referential aggregations* fulfill are the *safety* property described before and a *plausibleness* property for each group, meaning that the answer set represents a potential repair for each group, that consists in assigning to all the invalid references, the reference that represents each group.

For ascending aggregate functions, that is fr_count(*), fr_count() and fr_max(), each element of the aggregate lists is an upper bound of its group meaning that all the invalid references were updated with the reference represented by the element in question. For the descending function fr_min(), on the other hand, represents a lower bound. Also when the fr_sum() aggregate function behaves as an ascending/descending function, each element of the aggregate list is an upper/lower bound respectively.

The definition of full referential aggregations fr_count(*), fr_count(), fr_sum() is quite similar to the definition of our weighted referential aggregations. The only difference is a different computation for elements in the aggregate lists, which are computed without the $refw$ factor. We define the *full referential aggregation* for any aggregation agg() as

$$\mathcal{F}_{fr\_agg_{R_j.K}()}(R_i) = \mathcal{F}_{agg_{R_j.K}(fr\_agg())}({}_K\mathcal{F}_{fr\_agg_{R_j.K}()}(R_i))$$

where fr_agg() stands for the aggregate column in ${}_K\mathcal{F}_{fr\_agg_{R_j.K}()}(R_i)$. This definition also holds for the next aggregate functions.

We now define fr_max() (fr_min() is defined accordingly). For each correct reference k_c of foreign key $R_i.K$ that references $R_j.K$ the aggregate value is computed as follows:

$$\mathcal{V}({}_{K=k_c}\mathcal{F}_{fr\_max_{R_j.K}(A)}(R_i)) = max(\mathcal{V}({}_{K=k_c}\mathcal{F}_{max(A)}(R_i)), \mathcal{V}(\mathcal{F}_{max(A)}(R_i - \Pi_{R_i.*}(R_i \bowtie_K R_j))))$$

where $R_i.*$ stands for the list of attributes in R_i and function max() behaves as the SQL-max() aggregate in the presence of η. Observe fr_max() has the *consistency* property.

Full referential aggregate avg(), fr_avg() is defined as follows. In general, in SQL, avg() is not the same as $\frac{sum(A)}{count(K)}$. The function avg() computes the average of each row that qualifies. That is, in this case, rows that have a value in the A attribute different from η. The number of rows that qualify for the avg() function, not necessary matches the rows that qualify for count(K). A valid reference in attribute K associated with a η in A adds one to count(K) but not to the number of rows that qualify for avg(A). Observe that if both attributes are η, then both functions will not consider the tuple.

We now proceed with the definition of the elements of the aggregate list. For each correct reference k_c of foreign key $R_i.K$ that references $R_j.K$ the aggregate value is computed as follows:

$$\frac{\mathcal{V}({}_{K=k_c}\mathcal{F}_{fr\_sum_{R_j.K}(A)}(R_i))}{(|\sigma_{K=k_c \wedge notnull(A)}(R_i)| + |\sigma_{notnull(A)}(R_i - \Pi_{R_i.*}(R_i \bowtie_K R_j))|)}$$

where if the dividend expression is η and/or the divisor expression gives 0, then the quotient gives η. Finally, notice the *full referential aggregations* may also be improved following the same ideas presented in the last section.

4. EXPERIMENTAL EVALUATION

We conducted our experiments on a database server with one CPU running at 1 GHz with 256 MB of main memory and 108 GB on disk. Evaluation of aggregations were carried out on the public domain DBMS PostgreSQL.

4.1 TPC-H Database

Our synthetic databases were generated by the TPC-H DBGEN program [17], with scaling factors 1 and 2. We did not define any referential integrity constraint to allow referential errors. We inserted referential integrity errors in the

referencing fact table (*lineitem*) with different rates of errors (0.1%, 0.2%,..., 1%, 2%,..., 10%). The invalid values were inserted following several different probability distribution functions (pdfs) including uniform, geometric and normal, and in two foreign keys (*l_orderkey*, and *l_suppkey*).

The results we present in this section, use a default scale factor 1. The referencing table, *lineitem* and the referenced tables, *orders*, and *supplier* have the following cardinalities: 6M, 1.5M and 10k tuples, respectively. The invalid values were randomly inserted according to three different pdfs, that follow the parameters shown in Table 4. The minimum number of errors generated was approximately 6,000 and the maximum 600,000.

4.2 Approximation Accuracy

In order to evaluate the approximation accuracy for the *weighted referential aggregations*, we conducted the following experiment. We inserted referential integrity errors in the foreign key *l_suppkey* of referencing fact table *lineitem* with a 10% error rate. This was done following the three pdfs introduced above. Before doing so, we stored the valid references on another table in order to "repair" the invalid references when needed. We simulated a process of gradually repairing the database and within this process we also computed our proposed aggregate functions. Remember that in our framework, we are not interested in how repairs are done, but in getting an approximation of a complete answer set. We then evaluated the *weighted referential aggregations* and their corresponding standard SQL joined aggregations. Next, we repaired a 2% random subset of the original invalid references; our *weighted referential aggregations* and standard SQL joined aggregations were computed again. We repeated this process until the fact table was totally repaired. In each iteration we kept the aggregate values for each different group in order to compare such values with the "correct" ones on the final repaired table.

4.3 Time Performance

The queries used to compute the *weighted referential* and *full referential aggregations* first compute an auxiliary table. In SQL, this table is computed with a left outer join between the referencing table grouped by the foreign key and the referenced table and computes several aggregations depending on the function answer set that is needed. For example, for the wr_sum() exended aggregation it computes both count() and sum() for each group and the corresponding values for the invalid references. The next step is to compute the aggregate values of the invalid references, taken such references as a single group. The tuples in this group can be identified because the attribute that corresponds to the referenced primary key is η. Therefore, this group of invalid references can be constructed. These computations are done over the auxiliary table. The size of this table is the number of different values that are in the foreign key. Finally, the answer set is grouped by the grouping attribute.

We study the time performance of extended aggregations in Figure 6. for the aggregate functions wr_sum() and fr_sum(). Our experimental results evaluate performance of extended aggregation against standard SQL joined aggregations with foreign keys *l_orderkey* and *l_suppkey* of relation *lineitem*, with different rates of errors inserted as described before.

As we can see, there are even instances where our proposed aggregations perform better than the standard SQL joined aggregations. This is because: (1) an early aggregation grouping is computed before executing the join operation (push "group by" before join) and the remaining computations are done on the auxiliary table described earlier. For the sum() aggregations, performance depends on the size of the referenced table, as can be seen in Figure 6.

Summarizing, the performance of our extended aggregations computation depends on the size of the referencing table, the number of invalid values and the number of distinct values in the foreign key attribute.

PDF	Probability function	Parameters
Uniform	$\frac{1}{h}$	$h = 6000$
Geometric	$(1-p)^{n-1}p$	$p = 1/2$
Normal	$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	$\mu = 3000$ $\sigma^2 = 1000$

Table 4: PDFs used to insert invalid values.

5. RELATED WORK

Research on managing and querying incomplete data has received significant attention. In [6] the authors define a set of extended aggregate operations that can be applied to an attribute containing partial values. These partial values, which generalize the applicable null values [7], correspond to a finite set of possible values for an attribute in which only one of these values is the true one. The authors develop algorithms for several aggregate functions that deliver sets of partial values. In our work, we explore the idea of assuming that an incorrect reference represents imprecise data. The source of this value is an element of the set of valid references of the foreign key. This assumption, although strong, happens to be useful when we know the tuple holding the incorrect reference comes from a certain source database. Getting consistent answer sets from a query on an isolated database, where some integrity constraints are not satisfied is studied in [5]; the authors focus on time complexity and identify the set of inclusion dependencies under which getting a consistent answer set is decidable. In contrast, in our work we focus on aggregations on integrated databases, where referential integrity constraints are not satisfied. In [4] the authors identify two complementary frameworks to define views over integrated databases and they propose techniques to answer SPJ queries on integrated databases where there are missing foreign key values; the authors show that the problem of getting consistent answer sets is significantly difficult (non-polynomial time). Following a somewhat similar approach, in [2] the authors study scalar aggregation queries in databases that violate a given set of functional dependencies. They examine the problem of computing the range of all possible answer sets for aggregation queries, which results in a big search space. This approach has the benefit that, although the possible answer sets are incompletely represented by a range of values, the computations can be done in reasonable (polynomial) time. The authors do not address the specific problem of computing potential answer sets from aggregations in the presence of invalid foreign keys.

Uncertainty and imprecision have also been handled with extended data models that capture more information about the expected or probable behavior of data. By defining an imprecise probability data model [15], the authors can han-

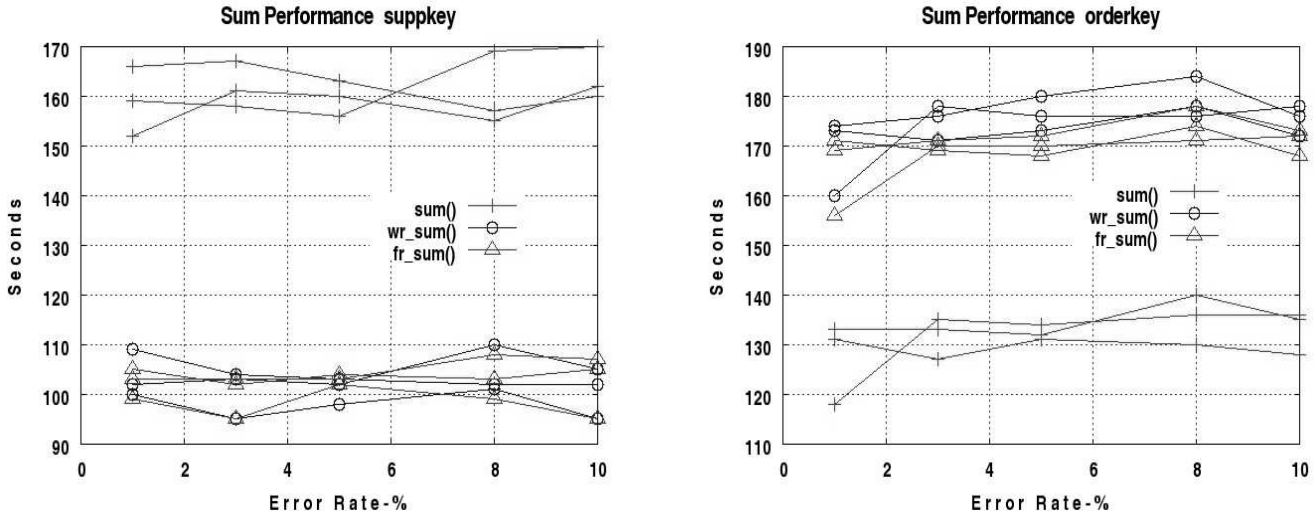

Figure 6: Comparing time performance of aggregations.

dle imprecise and uncertain data. They develop a generalized aggregation operator capable of determining a probability distribution for attributes with imprecise or uncertain values. They extend their method to cover aggregations involving several attributes. In our work we consider each invalid reference as a place holder (tag) where a crisp [15] but uncertain value should be stored. Also, associated to the values of the referenced primary key with respect to a given foreign key, there is a vector that holds for each value of the primary key, the probability that this value appears in a given tuple in the foreign key of the referencing relation. Power users may assign these probabilities, but we give a feasible method, exploiting the *weighted referential partial probability* vector, to compute such probabilities.

Reference [3] presents an extended OLAP data model to represent both uncertain and imprecise data. They introduced aggregation queries and the requirements that guided their semantics in order to handle ambiguous data. Certain knowledge about the data is needed to determine the probability that a fact has a precise value in an underlying possible world. We want to stress the fact that the authors do not discuss functionality when referential integrity errors occur. Such omission is important because in a data warehouse scenario, where accurate answer sets to aggregation queries are required, tables are likely to have referential integrity errors. Although in the approaches mentioned above consistent information with respect to the database and its assumptions is obtained, our proposal gives the user a data model and aggregate functions that handle erroneous data generated by integrity constraint violations and specifically by referential integrity errors.

6. CONCLUSIONS

We studied how to improve aggregations to return enhanced answers sets in the presence of referential integrity errors. Referential integrity errors are treated as imprecise data that stand for precise values, determined by a foreign key. We proposed two families of extended aggregate functions: weighted referential aggregations and full referential aggregations. Our extended aggregate functions are the counterparts of standard SQL aggregations. Weighted referential aggregations exploit a partial probability vector associated with the foreign key. Our proposal also includes a method to improve answer sets taking advantage of other related attributes via a functional dependency. Specifically, we analyze the case when such attributes are also foreign keys. Full referential aggregations present a potential repair scenario where each aggregated group receives all the values corresponding to existing referential integrity errors. These aggregations are helpful when the user needs to include for each aggregation group all tuples with invalid references or even tuples with invalid references that may probably belong to each group. Our extended aggregations exhibit important properties, which are required to consider them valid extensions of standard SQL aggregations. The weighted referential aggregation for row counts is consistent, ascending, and safe. The weighted sum aggregation is safe and consistent and when it behaves as an increasing or decreasing function, then it is ascending or descending, respectively. Full referential aggregations are safe and plausible. The latter property means the answer set represents a potential repair for each group, that consists in assigning to all invalid references, the reference that represents each group. Our experiments prove the answer sets returned by our extended aggregations are fairly accurate approximations and they also show the overhead due to additional computations is reasonable.

There are several issues for future work. Some of our ideas can be extended to general SPJ queries. We need to study query optimization of extended aggregations in more depth since sometimes they are slower than standard SQL joined aggregations. We need to consider other probability distribution functions for the partial probability vector, especially skewed distributions. We are exploring how to link partial probability vectors from two different attributes considering correlations. Unfortunately, weighted referential aggregations return answer sets that rarely correspond to actual answer sets given by potential repairs. Therefore, we want to improve the definition of weighted referential aggre-

gations to consider common repairs. We remain interested in the efficient computation of aggregations that return the most probable repaired answer sets.

Acknowledgments

The second author was sponsored by Macroproyecto de Tecnologías de la Información y la Computación from UNAM University. The second author would like to thank Hanna Oktaba and Sergio Rajsbaum, from UNAM University, for many interesting dicussions on this work.

7. REFERENCES

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *ACM SIGMOD Conference*, pages 34–48, 1987.

[2] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.*, 296(3):405–434, 2003.

[3] D. Burdick, P.M. Deshpande, T.S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP over uncertain and imprecise data. In *VLDB Conference*, pages 970–981, 2005.

[4] A. Cali, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. *Inf. Syst.*, 29(2):147–163, 2004.

[5] A. Cali, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *ACM PODS*, pages 260–271, 2003.

[6] A. L. P. Chen, J. S. Chiu, and F. S. C. Tseng. Evaluating aggregate operations over imprecise data. *IEEE TKDE*, 8(2):273–284, 1996.

[7] E.F. Codd. *The Relational Model for Database Management-Version 2*. Addison-Wesley, 1st edition, 1990.

[8] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *ACM SIGMOD Conference*, pages 240–251, 2002.

[9] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison/Wesley, Redwood City, California, 3rd edition, 2000.

[10] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-total. In *ICDE Conference*, pages 152–159, 1996.

[11] ISO-ANSI. *Database Language SQL-Part2: SQL/Foundation*. ANSI, ISO 9075-2 edition, 1999.

[12] A. J. Knobbe, A. Siebes, and B. Marseille. Involving aggregate functions in multi-relational search. In *PKDD02*, pages 145–168, 2002.

[13] H. J. Lenz and A. Shoshani. Summarizability in OLAP and statistical data bases. In *SSDBM*, pages 132–143, 1997.

[14] H. J. Lenz and B. Thalheim. OLAP databases and aggregation functions. In *SSDM*, pages 91–100, 2001.

[15] S. McClean, B. Scotney, and M. Shapcott. Aggregation of imprecise and uncertain information in databases. *IEEE TKDE*, 13(6):902–912, 2001.

[16] E. Rahm and D. Hong-Hai. Data cleaning: Problems and current approaches. *IEEE Bulletin of the Technical Committee on Data Engineering*, 23(4), 2000.

[17] TPC. *TPC-H Benchmark*. Transaction Processing Performance Council, http://www.tpc.org/tpch, 2005.

[18] J. Widom. Research poblems in data warehousing. In *ACM CIKM Conference*, pages 25–30, 1995.