Estimating and Bounding Aggregations in Databases with Referential Integrity Errors*

Javier García-García
Universidad Nacional Autónoma de México
Facultad de Ciencias
UNAM, Mexico City, CU 04510, Mexico
Carlos Ordonez
University of Houston
Department of Computer Science
Houston, TX 77204, USA

***Abstract*—Database integration builds on tables coming from multiple databases by creating a single view of all these data. Each database has different tables, columns with similar content across databases and different referential integrity constraints. Thus, a query in an integrated database is likely to involve tables and columns with referential integrity errors. In a data warehouse environment, even though the ETL processes take care of the referential integrity errors, in many scenarios this is generally done by including 'dummy' records in the dimension tables used to relate to the fact tables with referential errors. When two tables are joined, and aggregations are computed, the tuples with an undefined foreign key value are aggregated in a group marked as undefined effectively discarding potentially valuable information. With that motivation in mind, we extend aggregate functions computed over tables with referential integrity errors on OLAP databases to return complete answer sets in the sense that no tuple is excluded. We associate to each valid reference, the probability that an invalid reference may actually be a certain correct reference. The main idea of our work is that in certain contexts, it is possible to use tuples with invalid references by taking into account the probability that an invalid reference actually be a certain correct reference. This way, improved answer sets are obtained from aggregate queries in settings where a database violates referential integrity constraints.**

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems - *Query processing*

General Terms

Languages, Experimentation

Keywords

Functions, data quality, SQL

I. Introduction

There has been a growing interest on the problem of obtaining improved answer sets produced by queries in a setting where a database has incomplete content or violates integrity constraints [4], [3], [1], [2]. This is a common scenario in a data warehouse, where multiple databases of different reliability and similar contents are integrated. One way to deal with the problem is by updating the database in order to achieve consistency. This strategy has been thoroughly studied recently and several solutions have been proposed. In [5] consistency is achieved by inserting or deleting tuples, whereas in [16] attribute values are changed. The objective of the proposed techniques is to repair the original database to convert it into a consistent database. Once repaired, the database is ready to be exploited, for example, with OLAP queries. Fixing errors is difficult since it requires understanding inconsistencies across multiple tables, potentially going back to the source databases. In many data warehouse environments, the repair is done by adding rows to some dimension tables to make explicit that the fact with an invalid foreign key exists in the database but the dimension record is not available or undefined, thus leaving the database without referential interity errors.

In this work we propose a different strategy in order to obtain improved answer sets produced by queries posed over tables with invalid references that involve aggregation functions. Instead of repairing (changing) the invalid values of a foreign key in the original database, or inserting records into the dimension tables referenced by the fact tables with invalid values and returning "undefined" groups in the answer set, we estimate and bound aggregation answer sets by using the most likely values from the correct references. Our interest is to determine the expected correct values and the lower and upper bounds of aggregations where foreign keys are involved. In this paper we generalize and expand the ideas presented in [11]. Based on our initial studies, we present two families of extended aggregate functions and show that our aggregates together with the standard SQL grouped attribute aggregations computed over a joined table on foreign key-primary key attributes with potential referential integrity violations are part of a common probabilistic framework.

The article is organized as follows: Section II presents definitions. In Section III we explain how to compute aggregations in the presence of referential integrity errors and we introduce several families of extended aggregations. Section IV presents

*This is the authors version of the work. The official version of this article was published in Proc. ACM Workshop on Data Warehousing and OLAP (DOLAP, CIKM 2008 Workshop), p. 49-56, 2008

experiments. Section V discusses related research. Section VI concludes the article.

II. Definitions

A. Referential Integrity

A relational database is denoted by $D(\mathcal{R}, I)$, where $\mathcal{R}$ is a set of N tables $\mathcal{R} = \{R_1, R_2, \ldots, R_N\}$, R_i is a set of tuples and I a set of referential integrity constraints. A referential integrity constraint, belonging to I, between two tables R_i and R_j is a statement of the form: $R_i(K) \rightarrow R_j(K)$, where R_i is the referencing table, R_j is the referenced table, K is a *foreign key* (FK) in R_i and K is the primary key or a candidate key of R_j. In general, we refer to K as the primary key of R_j. To simplify exposition we assume simple primary and foreign keys and the common attribute K has the same name on both tables R_i and R_j.

Let $r_i \in R_i$, then $r_i\,[K]$ is a restriction of r_i to K. 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 $r_i \in R_i$ there must exist a tuple $r_j \in R_j$ such that $r_i\,[K] = r_j\,[K]$. The primary key of a table ($R_j.K$ 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 table are deleted and foreign keys are nullified.

Referential integrity can be relaxed. 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 $\mathcal{R}$. 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 $R_j.K$. In a data warehouse environment, a commonly used strategy to avoid that joins return answer sets that exclude tuples, is by inserting 'dummy' records into the dimension tables to explicitly indicate not available or undefined references.

B. Aggregations

Let $\mathcal{F}agg(R.A)$ be a simplified notation to denote the answer set returned by an aggregation, where agg() is an aggregate function and A is some attribute in R to compute aggregations on, or equivalently in SQL

SELECT $agg(R.A)$ FROM R.

The value of this atomic table with one tuple and one attribute will be denoted as $agg(R.A)$ to make it compatible for arithmetic and logical expressions. The aggregate function list over attribute A associated to the values of grouping attribute B will be denoted as ${}_B\mathcal{F}agg(R.A)$. Throughout the article, since there exist several different definitions for aggregate functions [9] which have distinct semantics, when we refer to an aggregate function agg(), it is taken from {count(*), count() and sum()} based on the standard SQL definition [6]. Our proposal can be applied in any database. However, our examples refer to an OLAP database. In this work, the following two tables will be used:

$R_i(\underline{PK}, \ldots, K, \ldots A, \ldots)$, $\quad R_j(\underline{K}, \ldots)$,

where R_i with primary key PK represents a referencing table playing the role of the fact table and R_j represents a referenced table 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 over which the aggregate function is applied.

We are particularly interested in computing aggregations over a joined table on foreign key-primary key attributes, with potential referential integrity violations, in this case, over $R_i \bowtie_K R_j$. We are also motivated by situations where the user is not interested in receiving an "undefined" group in the answer set as explained above.

Motivated by the fact that a null reference provides no information and the $\bowtie$ operator eliminates R_i tuples with a null on K, if $R_i.K$ in the referencing table is null in some tuple we may consider such tuple incorrect. However, for the cases where foreign keys are allowed to have nulls, as happens especially in data warehouses, less restrictive definitions are required that assume that foreign keys are allowed to have nulls. To consider both scenarios, and in order to simplify our exposition, throughout the article we will denote as $R[K]$ the set of values in $\pi_K(R)$ and may or may not include the null value depending on if it is considered valid or not. When null in the foreign key is considered correct, all the tuples with a null in its foreign key are treated as members of a single group of tuples, the null group. Proper explanations will be given for each case. We denote a generic null value by η.

With these ideas in mind, the answer set returned by an aggregation over a joined table on foreign key-primary key attributes, with potential referential integrity violations will be denoted as: $\mathcal{F}agg(R_i.A, r_i[K] \in R_j[K])$ or equivalently in SQL assuming η is invalid

SELECT $agg(R_i.A)$ FROM R_i JOIN R_j ON $R_i.K = R_j.K$

where agg() is an aggregate function, as defined above. The corresponding aggregate function list with grouping attribute K will be denoted as: ${}_K\mathcal{F}agg(R_i.A, r_i\,[K] \in R_j[K])$, written in SQL as

SELECT $agg(R_i.A)$
FROM R_i JOIN R_j ON $R_i.K = R_j.K$ GROUP BY $R_i.K$

For the cases where the user is not interested in receiving an "undefined" group in the answer set of the aggregation, to detect tuples with invalid references, we propose to create a view of the referenced table $R_j[K]$ that excludes the records that have undefined, not available or a similar description, causing the undefined values in $R_i[K]$ to be discarded from the answer set. These cases will be treated as if the undefined foreign key values were referential integrity errors.

In formal terms the problem we are solving is the following. An inconsistency may arise due to referential integrity errors when group:

$$ {}_K\mathcal{F}agg(R_i.A) \;\neq\; {}_K\mathcal{F}agg(R_i.A, r_i[K] \in R_j[K]) \quad (1) $$

and total aggregates:

$$ \mathcal{F}agg(R_i.A) \;\neq\; \mathcal{F}agg(R_i.A, r_i[K] \in R_j[K]) \quad (2) $$

are computed over joined tables. Finally, given $k \in R_j[K]$, we will denote as $agg(R_i.A, r_i\,[K] = k)$ the value of the aggregate function list ${}_K\mathcal{F}agg(R_i.A, r_i\,[K] \in R_j[K])$ that

sales

sId	cityId	amt	qtr
1	LAX	54	1
2	LAX	64	2
3	MEX	48	1
4	NXX	33	2
5	η	65	3
6	ROM	53	1
7	ROM	58	2
8	MAD	39	1

city

cityId	cityName
LAX	Los Angeles
LON	London
MAD	Madrid
MEX	Mexico
ROM	Rome

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

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

cityId	cityName	$sum(amt)$
LAX	Los Angeles	118
MAD	Madrid	39
MEX	Mexico	48
ROM	Rome	111
-TOT.	-	414

Fig. 2: Inconsistent answer set (total is inconsistent).

corresponds to the tuples where $r_i\,[K] = k$.

C. Motivating Example

Our examples throughout the article are based on the following database with two tables:
sales(sId, cityId, amt,qtr., ...), city(cityId, cityName, ...)

The 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. The referential integrity constraint,* sales(cityId) → city(cityId)*, should hold between the two tables. Now observe the query in Figure 2. The unioned query computes the sales amounts grouped by* city.cityId, cityName *and the total sales amount. But, as we can see, the answer set is inconsistent in a summarizable sense. That is, let* $s \in sales$*:* $\mathcal{F}sum(sales.amt) \neq \mathcal{F}sum(sales.amt, s[cityId] \in city[cityId])$ *their total sum of sales amount is different.*

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, 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. Invalid tuples are eliminated from the aggregate function answer set. Now, assuming that there is a high probability that the tuples with invalid foreign keys, that is, invalid values in attribute cityId*, represent true facts, could we improve the aggregate answer set grouped by* city.cityId*,* cityName*? Can we get an approximate answer set of the true real values?*

In a data warehouse environment, where no referential integrity errors exist due to the existence of a value in $city.cityId$ matching the undefined values in foreign key $sales.cityId$, $city$ is then a view of the original table with the record with the matching undefined value in $city.cityId$ discarded. The records with a foreign key value referencing the tuple marked as undefined, will be considered records with a referential integrity error.

III. Aggregations in Databases with Referential Integrity Errors

In this section we will present our proposal. First we will provide some preliminary definitions. Based on these definitions we present our extended aggregations in databases with referential integrity errors.

A. Preliminary Definitions

For the following definitions, consider a relaxed database where the referential integrity constraint $R_i(K) \rightarrow R_j(K)$ could be violated. As stated in Section II-B, we denote as $R_j[K]$ the set of values in $\pi_K(R_j)$ and may or may not include the null value and/or a 'dummy' value used to relate undefined values depending on if η (null) and/or the 'dummy' value are considered valid values or not.

The following definition is in the spirit of the definition of partial probability in [10]. A partial probability is a vector which associates a probability with each possible value of a partial value. This last value corresponds to a subset of elements of the domain of an attribute, and one and only one of the elements of the subset is the true value of the partial value.

Definition 1 Referential partial probability (RPP) *The RPP is a vector of probabilities that corresponds to a foreign key, say* $R_i.K$*, where its probabilities are associated to each value of the referenced primary key, say* $R_j.K$*. Each value corresponds to the probability that an invalid foreign key in* $R_i.K$ *is actually the associated correct reference in* $R_j.K$*. Let* $k \in R_j[K]$*, and* $p(k) \in RPP$ *the associated probability. We say that* $R_j.K$ *is complete under the RPP if*

$$\sum_{k \in R_j[K]} p(k) = 1 \tag{3}$$

The idea behind the RPPs is to associate to each primary key value 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 corresponding foreign key. Notice that for each referenced key, a set of RPPs may be defined, one or more for each foreign key that references it. Users with good database knowledge may assign these probabilities. Depending on the probabilities the user assigns to the valid foreign key values, different RPPs that satisfy completeness (Equation 3) can be defined. If the probability values are associated to a discrete probability distribution we can have a uniform or Zipf or geometric or in general, any probability distribution function RPPs. For example, if all the valid foreign key values were equally probable, a uniform RPP would be defined. Another special case could be a skewed probability distribution function where the user may want

to assign probability one to a specific valid reference and zero to all others. 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.

Definition 2 Frequency weighted RPP *Let $k \in R_j[K]$ and $n = |\{ r_i \in R_i \mid r_i[K] \in R_j[K] \}|$, the number of tuples with a valid reference in $r_i[K]$. We define $p(k)$ as*

$$p(k) = \begin{cases} \frac{|\sigma_{K=k}(R_i)|}{n} & if\ n \neq 0 \\ \\ \frac{1}{|R_j|} & otherwise \end{cases}$$

We are assuming a uniform probability distribution function if there are only invalid values in $R_i.K$, since we do not have more information. Thus defined then $R_j.K$ is complete under the *Frequency weighted RPP*.

Example 2 *Consider the relaxed database of Figure 1. The Frequency weighted RPP that corresponds to* sales.cityId *considering the values of the referenced primary key* city.cityId*,* ⟨LAX,MEX,ROM,MAD,LON ⟩*, is*

$$\langle \frac{2}{6}, \frac{1}{6}, \frac{2}{6}, \frac{1}{6}, 0 \rangle$$

Other feasible ways to assign the probabilities of the RPP achieving completeness can be designed such as a uniform or a constant RPP following the ideas presented above. The former one consists in associating to each correct reference an equal probability meaning that an invalid reference has an equal probability of being in fact any potentially valid reference. For this case, the value of each probability is $1/|R_j|$ or, if η is considered valid, $1/(|R_j|+1)$. The constant RPP consists in assigning to one potentially valid reference probability 1, meaning that all the invalid references are, in fact, the corresponding correct reference.

We can design RPPs that fail to meet completeness. Two special RPPs where completeness may not be satisfied are the following:

Definition 3 Full RPP *Define $p(k)$ as $p(k) = 1$.*

Definition 4 Restricted RPP *Define $p(k)$ as $p(k) = 0$.*

With the Full RPP we associate to each correct reference probability 1, meaning that every invalid value of the foreign key is in fact the associated valid value. On the other hand, with the Restricted RPP we associate probability 0 instead.

Definition 5 Referentiality (REF) *Let $r_i \in R_i$ and $k \in R_j[K]$, we define the referentiality of a foreign key value $r_i[K]$ with respect to k, $REF(r_i[K], k)$, as follows:*

$$REF(r_i[K], k) = \begin{cases} 1 & if\ r_i[K] = k \\ 0 & if\ r_i[K] \neq k \\ & \quad and\ r_i[K] \in R_j[K] \\ p(k) & if\ r_i[K] \notin R_j[K] \end{cases}$$

where the probability $p(k)$ corresponds to a given RPP.

TABLE I: Extended aggregates according to different RPPs.

Name	abrev.	prefix	RPP
Weighted referential	WR	w_	Any RPP that satisfies completeness
Frequency weighted referential	FWR	fw_	Frequency weighted
Full referential	FR	f_	Full
Restricted referential	RR	r_	Restricted

TABLE II: Referentialities (Definition 5) of foreign key *sales.cityId* values in valid tuples

$REF(s\,[cityId]\,, k)$	LAX	LON	MAD	MEX	ROM
< 1,*LAX*,..., 54,...>	1	0	0	0	0
< 2,*LAX*,..., 64,...>	1	0	0	0	0
< 3,*MEX*,..., 48,...>	0	0	0	1	0
< 6,*ROM*,..., 53,...>	0	0	0	0	1
< 7,*ROM*,..., 58,...>	0	0	0	0	1
< 8,*MAD*,..., 39,...>	0	0	1	0	0

Intuitively, REF($r_i[K]$,k) is the degree to which a foreign key value $r_i[K]$ in a tuple $r_i \in R_i$ refers to a correct reference $k \in R_j[K]$.

B. Extended Aggregate Function Definitions

For the following definitions, consider a relaxed database where the referential integrity constraint $R_i(K) \rightarrow R_j(K)$ could be violated. Let $r_i \in R_i$ and $k \in R_j[K]$. Our extendend aggregate functions computed over relaxed databases with referential integrity errors will be defined under a given RPP as follows:

$$x\_count(R_i.PK, r_i\,[K] = k) = \sum_{r_i \in R_i} REF(r_i\,[K]\,, k) \quad (4)$$

$$x\_count(R_i.A, r_i\,[K] = k) = \sum_{r_i \in R_i} REF(r_i\,[K]\,, k) \quad (5)$$

$$x\_sum(R_i.A, r_i\,[K] = k) = \sum_{r_i \in R_i} r_i\,[A] * REF(r_i\,[K]\,, k) \quad (6)$$

In Equations 5 and 6, we are assuming the tuples with η in $r_i[A]$ are ignored. For the x_sum() aggregates, we are assuming also, as in many OLAP scenarios (e.g. Example 1), that the $r_i[A]$ values, when different from zero, are always positive or negative. The specific name and meaning of the extended aggregate is obtained by changing prefix x_ and using the corresponding RPP, according to Table I.

Equation 4 corresponds to count(*). When η is assumed to be an invalid reference, the RR extended aggregates correspond to the standard SQL aggregations computed over a joined table on foreign key-primary key attributes, with potential referential integrity violations.

Example 3 *Consider the relaxed database of Figure 1. Let $s \in sales$. The referentiality of the values in* sales.cityId *that corresponds to each of the valid tuples is shown in Table II.*

The referentiality of the same foreign key that corresponds to the invalid tuples using different RPPs is shown in Table III. Next we show how to compute the different extended aggregates that correspond to the aggregate sum() *using the corresponding RPP shown in Table III.*

TABLE III: Referentialities (Definition 5) of foreign key *sales.cityId* values in invalid tuples with different RPPs (Definition 1).

$REF(s\,[cityId]\,,k)$	LAX	LON	MAD	MEX	ROM
Frequency weighted RPP					
$<4,\underline{\mathbf{NXX}},\ldots,33,\ldots>$	2/6	0	1/6	1/6	2/6
$<5,\ \underline{\eta}\ ,\ldots,65,\ldots>$	2/6	0	1/6	1/6	2/6
Full RPP					
$<4,\underline{\mathbf{NXX}},\ldots,33,\ldots>$	1	1	1	1	1
$<5,\ \underline{\eta}\ ,\ldots,65,\ldots>$	1	1	1	1	1
Restricted RPP					
$<4,\underline{\mathbf{NXX}},\ldots,33,\ldots>$	0	0	0	0	0
$<5,\ \underline{\eta}\ ,\ldots,65,\ldots>$	0	0	0	0	0
Uniform RPP					
$<4,\underline{\mathbf{NXX}},\ldots,33,\ldots>$	1/5	1/5	1/5	1/5	1/5
$<5,\ \underline{\eta}\ ,\ldots,65,\ldots>$	1/5	1/5	1/5	1/5	1/5

Frequency weighted referential:

$$fw\_sum(sales.amt, s\,[cityId] = LAX) = 1 \times 54 + 1 \times 64 + 0 \times 48 + 2/6 \times 33 + 2/6 \times 65 + 0 \times 53 + 0 \times 58 + 0 \times 39 = 150.66$$

Full referential:

$$f\_sum(sales.amt, s\,[cityId] = LAX) = 216.00$$

Restricted referential:

$$r\_sum(sales.amt, s\,[cityId] = LAX) = 118.00$$

Weighted referential:

$$w\_sum(sales.amt, s\,[cityId] = LAX) = 137.60$$

The last expression computed with the uniform RPP.

As for the definitions of the total aggregates, that is, the value of $\mathcal{F}x\_agg()$, using the simplified notation defined in Section II-B, we have the following:

$$x\_count(R_i.PK) = \sum_{k \in R_j[K]} x\_count(R_i.PK, r_i\,[K] = k) \quad (7)$$

$$x\_count(R_i.A) = \sum_{k \in R_j[K]} x\_count(R_i.A, r_i\,[K] = k) \quad (8)$$

$$x\_sum(R_i.A) = \sum_{k \in R_j[K]} x\_sum(R_i.A, r_i\,[K] = k) \quad (9)$$

C. Function Properties

Our extended aggregate functions must fulfill certain properties to be considered clean extensions of their counterpart standard SQL aggregations. The proofs of the following propositions are omitted due to lack of space.

Ascending/Descending, An *ascending* feature as defined in [7] holds for the aggregate functions x_count(*) and x_count(). That is, in our context, as tuples are inserted or deleted, the aggregate functions may increase (i.e. ascending) or decrease (i.e. descending). For x_sum() aggregate functions, there are cases where inserting or deleting tuples implies an increasing or decreasing aggregate as in many OLAP scenarios (e.g. Example 1), where the measure attribute, when different from zero, is always positive or negative. In these cases the aggregate functions x_sum() fulfill an *ascending* or *descending* feature.

Proposition 1 *The extended aggregates* x_count(*) *and* x_count() *are ascending aggregates. If* $\forall r_i \in R_i, r_i[A] \geq 0$ *then the* x_sum() *functions are ascending aggregates.*

Equivalently, for x_sum() with negative values in the measure attribute, we have the following:

Proposition 2 *If* $\forall r_i \in R_i, r_i[A] \leq 0$ *then the* x_sum() *functions are descending aggregates.*

Safety. If the referential integrity errors are repaired (in our context the referential integrity errors are repaired by the substitution of invalid references with correct references without varying the number of tuples) or if there are no referential errors, that is, if the referential integrity constraint holds for all tuples, a *safety* feature holds for the extended aggregations, meaning that the answer sets will not be different compared to the ones from the standard SQL joined aggregations, that is, the SQL grouped attribute aggregations computed over a joined table on foreign key-primary key attributes. Here, we assume η is invalid.

Proposition 3 *If* $\forall r_i \in R_i, r_i[K] \in R_j[K]$ *then* $\mathcal{F}x\_agg() = \mathcal{F}agg()$.

Summarizable consistency. For the WR and FWR count(*), count() and sum() aggregate functions, a *summarizable consistency* property holds. That is, these distributive aggregate functions applied to an attribute is equal to a function applied to aggregates, that, in turn, are generated by the original aggregate function applied over the attribute of each partition of the table. This property corresponds to the *summarizability* feature described in [8]. That is, a distributive function over a set should preserve the results over the subsets of its partitions. The proof is based on the definition of referentiality, Definition 5, and the completeness property of the RPP of these type of aggregates, Equation 3.

Proposition 4 *Let* $r_i \in R_i$ *and attribute* PK *its primary key. Then*

$$w\_count(R_i.PK) = \sum_{k \in R_j[K]} w\_count(R_i.PK, r_i\,[K] = k) = \mid R_i \mid$$

Summarizable consistency for w_sum($R_i.A$) can be formulated as: $w\_sum(R_i.A) = \sum_{k \in R_j[K]} w\_sum(R_i.A, r_i\,[K] = k) = \sum_{r_i \in R_i} R_i.A$.

WR and FWR count(*), count() and sum() total aggregates are invariant wrt referential integrity repairs. As the referential integrity errors are repaired, the total aggregate remains *invariant wrt referential integrity repairs*. That is, the total aggregate remains constant during this type of repair processes.

The elements of the aggregate lists of the RR and FR extended aggregates are plausible. A *plausibleness* property means that the answer set represents a potential repair of the table. For the FR aggregates, the repair consists in assigning to all the invalid references, the valid reference we are considering. For the RR aggregates, this repair consists in never updating the invalid reference with the valid value we are considering.

D. Probabilistic Interpretation

In order to obtain a valid inference from our extended aggregate functions, it is important that the user bears in mind the following assumptions. Notice we are assuming that $R_j[K]$ is complete. That is, the set of referenced values

are all the possible valid values, possibly with the η value, depending on if it is considered valid or not. On the other hand, attribute $R_i.K$ is assumed to have potentially invalid references. Alternative approaches, may consider $R_i.K$ invalid references valid after all, assuming that the error is due to an incomplete set of references in $R_j[K]$.

Users may assign different RPPs, nevertheless, the Frequency weighted RPP assumes that the probability that a certain foreign key valid value be the actual value that should stand instead of the invalid value in a foreign key depends on the occurrence, frequency, of that same valid value in the given foreign key. That is, the occurrence is not completely random, it depends on the observed valid values. On the other hand, we are assuming also that the occurrence of an invalid value does not depend on the invalid values. As for the aggregate functions like sum() where the aggregate function is applied over an attribute we are assuming that the foreign key and the invalid values are not related to the attribute in question. That is, the values of the attribute do not depend on the values of the foreign key.

Now consider a set of binomial random variables each one of them represented by a potentially valid reference of a given foreign key $R_i.K$. Suppose each random variable has as its initial value the number of tuples where the value it represents is present in $R_i.K$. Next, given our assumptions, suppose that each tuple of R_i with an invalid reference in attribute K is an independent trial of a given random variable, say the one represented by the potentially valid reference $k \in R_j[K]$. Let the probability of success of the binomial random variable represented by k be the corresponding probability in the RPP. A successfull trial, in this context, represents the fact that an invalid reference is updated with value k.

Notice that if $k \in \pi_K(R_i)$ (if η is valid, then it should be considered also) then $|\{r_i \mid r_i \in R_i \ \wedge \ r_i[K] = k\}|$, in our context, is the lower bound of the corresponding binomial random variable. If $k \notin \pi_K(R_i)$, then the lower bound is 0, that is, no tuples with the valid reference k in $r_i[K]$. In both scenarios, the lower bound represents the case where all the trials (tuples with referential integrity errors) were unsuccessful. That is, the case where the actual value of attribute $R_i.K$ in the invalid tuples is different from k. This number corresponds to the correct tuples with $r_i[K] = k$. The upper bound of this binomial random variable represents the case that all the tuples with referential integrity errors were successful. That is, the case where all the actual values of the invalid values of foreign key $R_i.K$ are indeed k. We can see then that for the random variables described above, if $k \in \pi_K(R_i)$, the probability is 0 that it takes a value lower than $|\{r_i \mid r_i \in R_i \ \wedge \ r_i[K] = k\}|$, once the invalid references are repaired and the probability is 1 that it has a value lower or equal to $|\{r_i \mid r_i \in R_i \ \wedge \ r_i[K] = k\}| + |\{r_i \mid r_i \in R_i \ \wedge \ r_i[K] \notin R_j[K]\}|$ once the repair process takes place. If $k \notin \pi_K(R_i)$ the corresponding values are 0 and $|\{r_i \mid r_i \in R_i \wedge r_i[K] \notin R_j[K]\}|$ respectively. Now observe we can compute the expected value of the binomial random variable by adding to its initial value the product between the number of independent trials (invalid tuples) and its corresponding probability in the RPP. This value represents the expected number of tuples in R_i that will eventually end with value k in attribute K.

Following these ideas, we can see that the RR and FR variants of aggregates x_count(*), x_count() and x_sum() are the lower and upper bounds, respectively, of the value the corresponding standard aggregate may take when the referential integrity errors are repaired. The WR and FWR are the expected value, again, of the corresponding standard aggregates and its result depends on which RPP is considered.

E. Discussion

In order to evaluate the usefulness of the answer sets delivered by the WR and the FR aggregations the following important aspects have to be discussed: How hard is to compute all the plausible answer sets of the aggregate functions?, how many are there?, and does a repair process will eventually give the answer set delivered by the weighted referential aggregations? We discuss these issues below.

Let e be the number of tuples in R_i with a referential integrity error in attribute K, that is, $e = |\{r_i | r_i \in R_i \wedge r_i[K] \notin R_j[K]\}|$. Also let a *referential integrity repair of attribute* $R_i.K$ be a new instance of R_i, with the same number of tuples, but with the invalid values of attribute K replaced with valid values taken from the set of values of $R_j[K]$. The number of potential referential integrity repairs of attribute $R_i.K$ is $(\mid R_j[K] \mid)^e$. For our example in Figure 1 there are 25 potential referential integrity repairs of attribute $sales.cityId$ which is a big number considering there are only 2 referential integrity errors. As for the number of plausible values of a given group once a repair process of a given foreign key have taken place, given the interpretation just discussed we can see that for the aggregate function count() there are $e + 1$ or less plausible answers and 2^e at most for the aggregate function sum(). For our example in Figure 1 take the group represented by the value LAX. The plausible answers for the aggregate function count() for the group represented by value LAX are $\{2, 3, 4\}$ since there are 2 invalid references. The aggregation fw_count() gives us 2.6 for value LAX since there are 2 valid tuples with this value, the total number of errors is 2 and, as we saw in Example 2, the probability of value LAX in the corresponding RPP is $2/6$, considering η as an invalid reference. If we compute the probabilities of each of the plausible answers considering the RPP of the same example we have $\{(2, 0.44), (3, 0.44), (4, 0.11)\}$, where the first number of each pair is the plausible answer and the second its probability. The cumulative probability of the plausible answer 3 is 0.88 with the plausible answers sorted in ascending order, meaning that the probability is 0.88 that the answer be 3 or less once a repair process of foreign key *sales.cityId* takes place.

In the same way, for the aggregate function sum() the plausible answers for value LAX and their corresponding probabilities considering the same RPP as above, are $\{(118, 0.44),\ (151, 0.22), (183, 0.22), (216, 0.11))\}$. The cumulative probability of the plausible answer 151 is 0.66 with the plausible answers sorted on ascending order. As we can see from Example 3 the corresponding value of the fw_sum() for value LAX, *Los Angeles*, is 150.66.

TABLE IV: Transaction detail in a given point of sale (p. of s.) for a given sellerId.

transId	clientId	agentId	prodId	amt	sellerId
1276	143547	45779	814	$22,347.83	9343
1277	143547		821	$16,086.96	9343
1278	243577	17425	677	$1,477.39	9343
1279	193430	17378	684	$2,826.09	9343
1280	192430		225	$773.91	9343
1281	289940		218	$513.04	9343
1282	268948		784	$4,513.04	9343

TABLE V: Total commission per agent

concept	amt
Total sales amt.	$1′654,404
Total sales amt. without agent	$110,001
Commission paid	$154,440
Bonus paid	$11,000

We can see then that the proposed aggregations are a very efficient way to compute the estimated answer sets and the upper and lower bounds of the corresponding aggregate functions, although we do not pretend to give an exact result of a repair process.

IV. Experimental Evaluation

We used a real and a synthetic database, generated with the TPC-H DBGEN program [15]. We used standard SQL (ANSI) in our implementation, making it portable in each relational DBMS.

A. Real Database

An important retail company in Mexico listing at the stock market since more than 25 years ago, tried our extended aggregates in one of its applications to assess the usefulness of our approach. Our aggregates were used in a reward program applied to agents. The agents are advisers working in specialized departments that participate in sales and they earn commissions based on sales depending on the number of sales and the total amount sold of their corresponding products. In every point of sale, a seller records the information related to a sale including the agent's code, but in several occasions this code is omitted or is erroneous, since this particular data is manually inputted. The company has separated file systems in several stores nationwide and the information is daily concentrated in a centralized database. In this database, about 7% of the total number of sales that should appear with an agent's code, have an invalid value in this field. In Table IV are several records showing how the information is received, some of them with no information in the agent's code field, *agentId*.

Several assumptions about the database were discussed and were validated by the user in order to obtain valid inferences from the extended aggregates, for example: a referential integrity error did not depend on the particular type of transaction nor on the attribute that was aggregated.

The commission is paid after a given time to avoid paying an agent when a product is returned. A bonus was added to compensate the sales that were inputted without a valid value in *agentId*. This bonus is computed using the fw_sum() aggregate considering the total amount of sales without an agent code and taking into account the total number of sales where an agent took part. Tables V and VI show the amount of sales per agent, the commission earned and the bonus computed using fw_sum().

TABLE VI: Bonus computed using fw_sum()

date	agentId	amt.	comm.	sales	bonus
02/23/07	45779	$282,231	$28,223	12	$1,760
02/23/07	17425	$438,111	$43,811	13	$1,906
02/23/07	17378	$138,168	$138,16	9	$1,320
03/14/07	84536	$333,949	$33,394	17	$2,493
03/14/07	13754	$171,440	$17,144	14	$2,053
03/14/07	17033	$180,504	$18,050	10	$1,466

B. TPC-H Database

Our synthetic databases were generated by the TPC-H DBGEN program with scaling factors 1 and 2. We inserted referential integrity errors in the referencing 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, Zipf and geometric, and in two foreign keys ($l\_orderkey$, and $l\_suppkey$).

The results we present in this section, use a default TPC-H 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.

Approximation Accuracy: In order to evaluate the approximation accuracy for the WR aggregations, we conducted the following experiments. We inserted referential integrity errors in the foreign key $l\_suppkey$ of referencing table $lineitem$ with a 10% error rate. The erroneous values were generated so that they follow the three pdfs mentioned above and were inserted randomly in order to simulate a scenario where the errors occurred in an independent manner. 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. We then evaluated the FWR aggregations and their corresponding standard SQL joined aggregations. Next, we repaired a 2% random subset of the original invalid references; our FWR aggregations and standard SQL joined aggregations were computed again. We repeated this process until the 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.

Figure 3 shows the accuracy of fw_sum() with the measure attribute $l\_extendedprice$ in table $lineitem$ assuming a Zipf pdf. We show the maximum and minimum correct aggregate values eventually reaching their corresponding values where the error rate is 0%. As we can see, the lines that correspond to the fw_sum() values are almost constant (horizontal line), meaning that the estimated values become increasingly similar to the final real aggregate value. The function fw_sum() converges to the standard SQL joined aggregation sum().

Time Performance: We study the time performance of extended aggregations in Figure 4 for the aggregate func-

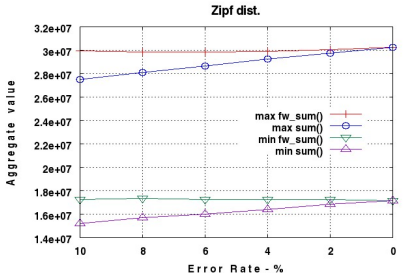

Fig. 3: Accuracy of the fw_sum() aggregate function.

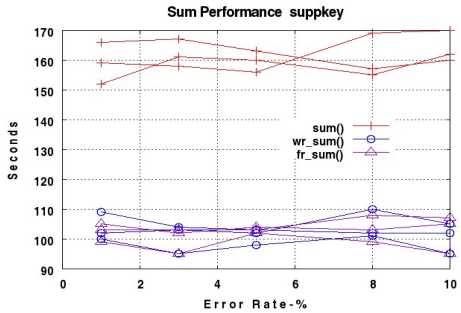

Fig. 4: Comparing time performance of aggregations.

tions fw_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 table $lineitem$, with different rates of errors inserted as described before. In general, time performance is good, slightly slower than SQL.

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 an auxiliary table whose size depends on the number of different foreign key values. For the sum() aggregations, performance depends on the size of the referenced table. For the WR aggregates, the additional computations are done over the auxiliary table. This overhead is linear in the size of the referenced table.

V. Related Work

Research on managing and querying incomplete data has received significant attention. In [4] the authors define a set of extended aggregate operations that can be applied to an attribute containing partial values. These partial values, which generalize applicable null values, 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 a similar idea, 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 specific source database. In [1] the authors study scalar aggregation queries in databases that violate a given set of functional dependencies. They study the problem of computing the ranges of all possible answer sets for aggregation queries, which results in a big search space.

By defining an imprecise probability data model [10], the authors can handle 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 [10], but uncertain value should be stored.

Concerning aggregate operators in probabilistic databases in [14] the authors define aggregate operators over probabilistic DBMSs and present linear programming based semantics for computing these aggregate operators. They present approximation algorithms that run in polynomial time, but the result may be an approximation of the correct answer. An important difference with our work is that the aggregate operators in probabilistic databases are defined over probability intervals. The use of a RPP to assign a single probability to each invalid foreign key is a key element to the efficiency of our proposed aggregates.

To close our discussion we summarize past research on improving database systems to handle referential integrity issues. In [12] we propose to measure referential integrity errors. We introduced the early foreign key grouping optimization technique mentioned in Section IV. In [13] we extended the ideas to distributed databases. In [11] we presented an initial study of how to improve aggregations.

VI. Conclusions

We improved SQL aggregations to return enhanced answers sets in the presence of referential integrity errors. Referential integrity errors are treated as imprecise values that stand for precise values, determined by a foreign key. We proposed two families of aggregate functions: weighted referential (WR) aggregations and full referential (FR) aggregations. These aggregations represent a complement to standard SQL aggregations and they are studied under a common probabilistic framework. WR aggregations are based on referential partial probability vectors (RPPs) associated with the foreign key. Full referential aggregations are helpful when the user needs to include for each group all tuples with invalid references. Our experiments show answer sets returned by extended aggregations are consistent approximations and they also show the overhead due to additional computations is reasonable.

Acknowledgments

The first author was sponsored by the UNAM IT project "Macroproyecto de Tecnologías para la Universidad de la Información y la Computación".

References

[1] 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.

[2] 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.

[3] A. Calì, 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.

[4] 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.

[5] G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE TKDE*, 15(6):1389–1408, 2003.

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

[7] A. J. Knobbe, A. Siebes, and B. Marseille. Involving aggregate functions in multi-relational search. In *PKDD02*, pages 287–298, 2002.

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

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

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

[11] C. Ordonez and J. García-García. Consistent aggregations in databases with referential integrity errors. In *ACM IQIS*, pages 80–89, 2006.

[12] C. Ordonez and J. García-García. Referential integrity quality metrics. *Decision Support Systems Journal*, 44(2):495–508, 2008.

[13] C. Ordonez, J. García-García, and Z. Chen. Measuring referential integrity in distributed databases. In *ACM CIMS*, pages 61–66, 2007.

[14] R. Ross, V.S. Subrahmanian, and J. Grant. Aggregate operators in probabilistic databases. *J. ACM*, 52(1):54–101, 2005.

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

[16] J. Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3):722–768, 2005.