Extended Aggregations for Databases with Referential Integrity Issues *

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

Querying databases with incomplete or inconsistent content remains a broad and difficult problem. In this work, we study how to improve aggregations computed on databases with referential errors in the context of database integration, where each source database has different tables, columns with similar content across multiple databases, but different referential integrity constraints. Thus, a query in an integrated database may involve tables and columns with referential integrity errors. In a data warehouse, even though the ETL processes fix referential integrity errors, this is generally done by inserting "dummy" records into the dimension tables corresponding to such invalid foreign keys, thereby artificially enforcing referential integrity. When two tables are joined and aggregations are computed, rows with an invalid or null foreign key value are skipped, effectively eliminating potentially valuable information. With that motivation in mind, we extend SQL aggregate functions computed over tables with referential integrity errors to return complete answer sets in the sense that no row is excluded. We associate to each referenced key in the dimension table, a probability that invalid or null foreign keys refer to it. Our main idea is to compute aggregations over joined tables including rows with invalid or null references by distributing their contribution to aggregation totals, based on probabilities computed over correct foreign keys. Therefore, our extended aggregations can return improved answer sets in databases that violate referential integrity or have referential issues. Experiments with real and synthetic databases evaluate the usefulness, accuracy and performance of our extended aggregations.

Email addresses: javgar@servidor.unam.mx (Javier García-García), ordonezc@cs.uh.edu (Carlos Ordonez)

This is the author's version. Official version published in Elsevier DKE, 69(1):63-95, 2010

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 has incomplete content or violates integrity constraints [9, 10, 8, 2, 4, 1]. This is a common scenario in a data warehouse, where multiple databases of different reliability and similar contents are integrated. Databases with referential integrity errors commonly arise in scenarios where several organizations have their databases integrated, where exchanging or updating information is frequent, or where table definitions change. Database integration represents a common scenario where similar tables coming from multiple source databases (OLTP systems) have different referential integrity constraints and each DBMS provides distinct mechanisms and rules to enforce referential integrity. Therefore, source databases may violate referential integrity and their integration may uncover additional referential integrity problems. Another scenario where referential integrity problems arise is when, because of performance reasons referential integrity checking is disabled to allow fast insertions in batch mode. This action may cause the appearance of referential integrity errors. Finally, the logical data model behind a relational database evolves, incorporating new attributes and new relations not defined before, causing old data to violate new referential integrity constraints.

One way to deal with the problem of violation of integrity constraints is by updating the database in order to achieve consistency. This strategy has been thoroughly studied recently and several solutions have been proposed. In [19, 3] consistency is achieved by inserting or deleting tuples, whereas in [36] 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. A good overview on data cleaning can be found in [14] and [32]. This strategy presents several disadvantages. First of all, in many real-world scenarios, we cannot be sure that the techniques used to repair the database are error-free. The chief disadvantage about such approach is that the database must be modified. Second, the original database is updated and the user loses track of which data elements represent either repaired data or correct data. Third, another solution to the problem is to repair the database by removing inconsistent data. Removing data is the easiest, but generally not an acceptable solution. 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 integrity errors. If a valid fact record has an invalid foreign key value, a record in the associated dimension is created with a value of 'undefined', and during the ETL process, the invalid foreign key value in the fact table is changed to reference this record. In a data warehousing environment it is essential to repair referential integrity errors as early as possible in the ETL process, as it is recommended in [22]. Letting referential errors go undetected can lead to

expensive repair processes and queries producing incomplete answers. When a data warehouse has many denormalized tables (materialized views) repairing referential integrity can become prohibitively expensive. A common strategy to repair the referential integrity errors is to substitute the invalid references with a "valid one" that refers to a tuple in the dimension table that is marked explicitly as undefined or not available (e.g. NA). This solution, although it repairs the referential error, does not help much if the user wants to estimate or bound particular aggregated groups of the answer sets of aggregate functions.

In this work we propose an innovative strategy in order to obtain improved answer sets produced by queries posed over tables with records with undefined references or with referential integrity errors that involve aggregation functions. Instead of repairing (changing) the invalid values of a foreign key in the original database, or returning "undefined" groups in the answer set by aggregating the records in the fact tables with an "undefined" value in the aggregated foreign key, we estimate and bound aggregation answer sets by using the most likely values from the correct references. We introduce extended aggregate functions computed over databases with undefined references or with referential integrity errors. In contrast with the repair techniques mentioned above, we are not interested in updating the database foreign key values by substituting invalid specific data with valid data. Our interest is to define extended aggregate functions that dynamically and efficiently determine the expected correct answer set or determine the lower and upper bounds of the answer set of standard aggregates computed over a joined table on foreign key-primary key attributes with potential referential integrity violations or with undefined references. In this paper we generalize and expand the ideas presented in [29] and [18]. Based on our initial studies, we present a probabilistic interpretation of the extended aggregate functions and show that these functions 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.

This is an outline of the rest of our article. Section 2 presents definitions and motivating examples. Section 3 presents our main contributions. We explain how to compute aggregations in the presence of undefined references or of referential integrity errors and we introduce several families of extended aggregations. We show how we can implement our proposal in SQL. We show how the extended aggregates may be improved in the presence of columns not independent. Section 4 presents experiments with real and synthetic databases. Section 5 discusses related research. Section 6 concludes the article.

2. Definitions

2.1. 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 [15]. We refer to the valid state just defined as a *strict state*. In a data warehouse environment, a commonly used strategy to avoid joins that return answer sets with excluded tuples is by inserting records into the dimension tables to explicitly indicate not available or undefined references. When two tables are joined and aggregations are computed, the tuples with an undefined foreign key value that reference the undefined dimension tuple are aggregated in a group marked as undefined. Since this strategy effectively discards potentially valuable information, we also define a *rigorous state*. A rigorous state is a strict state where all the dimension tuples are defined, as opposed to the undefined dimension tuples explained above.

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 $R_j.K$.

2.2. 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 ${}_B\mathcal{F}agg(R.A)$, or in SQL

```
SELECT agg(R.A)
FROM R
GROUP BY R.B;
```

Throughout the article, since there exist several different definitions for aggregate functions [26] which have distinct semantics, when we refer to an aggregate function agg(), it is taken from {count(*), count(), sum(), max() and min()} based on the standard SQL definition [21].

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:

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

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, in this case over $R_i \bowtie_K R_j$, with potential referential integrity violations, or with undefined references created to preserve referential integrity. That is, over relaxed or strict databases. We are motivated by situations where the user is not interested in receiving an undefined group in the answer set as explained above but interested in receiving an answer set a rigorous database would provide.

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. Appropriate explanations will be given for each case. We denote a generic null value by η.

Throughout the article, since we are assuming that the data warehouse user is not interested in receiving an undefined group in the answer set of an aggregation over a joined table on foreign key-primary key attributes, $R_j[K]$ will always represent a view of the referenced table R_j that will exclude the records that have undefined, not available or a similar description, $R_j[K] = \pi_K(\sigma_{K \neq' NA'}(R_j))$, causing the undefined values in $R_i[K]$ to be discarded from the answer set. These cases will be treated as if the foreign key values pointing the dimension tuples explicitly marked as undefined were referential integrity errors.

With these ideas in mind, the answer set returned by an aggregation over a joined table on foreign key-primary key attributes 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 $R_j[K]$ 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 with the same assumption as above as

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

In formal terms the problem we are solving is the following. An inconsistency may arise due to undefined references or 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 corresponds to the tuples where $r_i[K] = k$. This is a convenient shorthand for the value that corresponds to the answer set given by the following equivalent expression written in SQL to highlight the selected group

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

Equivalent SQL expressions are given throughout our work since our proposal was implemented and evaluated in SQL.

2.3. Motivating Examples

Our examples throughout the article are based on a chain of stores database with three tables:

sales(storeId, cityId, regionId, amt,qtr., . . .)
city(cityId, cityName, country, . . .)
region(regionId, regionName, . . .)

which are the result of integrating two databases from two companies, X and Y, that are in a process of database integration. X had store information organized by city and Y 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, assuming that the null value is invalid.

sales

storeId	cityId	regionId	amt	qtr	...
1	LAX	AM	54	1	...
2	LAX	AM	64	2	...
3	MEX	AM	48	1	...
4	**NXX**	AM	33	2	...
5	η	AM	65	3	...
6	ROM	EU	53	1	...
7	ROM	EU	58	2	...
8	MAD	EU	39	1	...

city

cityId	cityName	country	...
LAX	Los Angeles	US	...
LON	London	UK	...
MAD	Madrid	SP	...
MEX	Mexico	MX	...
ROM	Rome	IT	...

region

regionId	regionName	...
EU	Europe	...
AM	Americas	...

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

sales

storeId	cityId	regionId	amt	qtr	...
1	LAX	AM	54	1	...
2	LAX	AM	64	2	...
3	MEX	AM	48	1	...
4	**NA**	AM	33	2	...
5	**NA**	AM	65	3	...
6	ROM	EU	53	1	...
7	ROM	EU	58	2	...
8	MAD	EU	39	1	...

city

cityId	cityName	country	...
LAX	Los Angeles	US	...
LON	London	UK	...
MAD	Madrid	SP	...
MEX	Mexico	MX	...
ROM	Rome	IT	...
NA	Not Available	η	...

region

regionId	regionName	...
EU	Europe	...
AM	Americas	...

Figure 2: The same store database shown in Figure 1 in a strict state, after an ETL process, with the invalid references changed to point to a tuple explicitly marked as undefined.

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

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

Figure 3: Inconsistent answer set (total is inconsistent).

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

sum(amt)
316

Figure 4: Total sales from valid references (total is inconsistent).

Figure 2 shows a strict database with no referential integrity errors, which is the same database as above but after an ETL process where the invalid references **NXX** and η in table *sales* were changed to point to a new tuple in table *city* with the following data: $< NA, Not\ Available, \eta, \ldots, >$.

Example 1 *Take the database in Figure 1. The attribute* cityId *in* sales *is a foreign key. The referential integrity constraint,* sales(cityId) $\rightarrow$ city(cityId)*, should hold between the two tables. Now observe the query in Figure 3 computed over this database. The unioned query computes the sales amounts grouped by* city.cityId, cityName *and the total sales amount. But, as we can see from the query in Figure 4, 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. With a strict database obtained after fixing the referential integrity errors with undefined references as the one in Figure 2, the answer set would contain a tuple with a group marked as Not Available*. Although the answer set would be consistent, no hint could we have about the real aggregate values of the real cities. Now, assuming that there is a high*

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

regionId	regionName	$sum(amt)$
AM	Americas	264
EU	Europe	150
-TOTAL	-	414

Figure 5: Total sales from all valid references on another FK (total is consistent).

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?*

Example 2 *Based on the database integrity constraints, the functional dependency:* sales.cityId → sales.regionId*, should hold. Suppose the information from Y is more reliable than information from X. A query getting total sales by region and total overall sales is shown in Figure 5. In this case, a summarizable 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* city.cityId, cityName *in the presence of invalid or undefined foreign key values? This will be a goal of our approach.*

3. Aggregations in Databases with Referential Integrity Issues

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 data warehouses with undefined references or databases with referential integrity errors.

3.1. 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 or a strict database where the referential integrity errors were repaired by adding a tuple in the referenced table, explicitly marked as undefined, and setting the referential integrity errors of the referencing table to refer to this tuple. As stated in Section 2.2, we denote as $R_j[K]$ the set of values in $\pi_K(R_j)$ excluding the records that have undefined, not available or a similar description.

The following definition is in the spirit of the definition of partial probability in [27]. 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 defined value of the referenced primary key, say $R_j[K]$. Each value corresponds to the probability that an invalid or an undefined foreign key reference in $R_i.K$ is actually the associated correct defined 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 referenced, defined 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 or an undefined 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 defined foreign key values, different RPPs under which completeness may be satisfied (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, defined 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 $n = \mid \{\ r_i \in R_i \mid r_i[K] \in R_j[K]\ \} \mid$, the number of tuples with a valid, defined reference in $r_i[K]$. We define $p(k)$ as*

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

We are assuming a uniform probability distribution function if there are only undefined or 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 3 *Consider the relaxed database of Figure 1 or the strict database of Figure 2. The Frequency weighted RPP that corresponds to* sales.cityId *considering the defined values of the referenced primary key* city.cityId, ⟨LAX,MEX,ROM,MAD,LON ⟩*, is*

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

Here, we are assuming η is an invalid value. If this were not the case, the Frequency weighted RPP that would correspond to the correct values taking η as valid, that is, for ⟨LAX,MEX,ROM,MAD,LON, η ⟩ *the Frequency weighted RPP would be*

$$\langle \frac{2}{7}, \frac{1}{7}, \frac{2}{7}, \frac{1}{7}, 0, \frac{1}{7} \rangle$$

Notice in Figure 2 that if we assume η is a valid value, the ETL process would leave η in attribute sales.cityId *in tuple* 5.

We can design RPPs that fail to meet completeness. Two special RPPs where completeness may not be satisfied are the following: For the next two definitions, let $k \in R_j[K]$

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 defined, correct reference probability 1, meaning that every undefined or invalid reference 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 $k \in R_j[K]$, a defined reference, we define the referentiality of a foreign key value $r_i[K]$ with respect to k, $REF(r_i[K], k)$, as follows:*

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

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

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 $k \in R_j[K]$.

Table 1: Extended aggregates according to different RPPs (Definition 1).

Name	**abrev.**	**prefix**	**RPP**
Weighted referential	WR	w_	any RPP under which completeness is satisfied
Frequency weighted referential	FWR	fw_	Frequency weighted
Full referential	FR	f_	Full
Restricted referential	RR	r_	Restricted

3.2. 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 or a strict database with undefined references as described in Section 2.1. Let $r_i \in R_i$ and $k \in R_j[K]$ be a defined correct reference value. Our extendend aggregate functions computed over relaxed databases with referential integrity errors or strict databases with undefined references will be defined under a given RPP as follows:

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

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

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

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 1.

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 4 *Consider the relaxed database of Figure 1 or the strict database of Figure 2. Let $s \in sales$. The referentiality of the values in* sales.cityId *with respect to $k \in city[cityId]$ that corresponds to each of the tuples with defined references is shown in Table 2.*

The referentiality of the same foreign key that corresponds to the tuples with invalid or undefined references using different RPPs is shown in Table 3.

Next we show how to compute the different extended aggregates that correspond to the aggregate sum() *using the corresponding RPP shown in Table 3.*

Table 2: Referentialities (Definition 5) of foreign key *sales.cityId* values in valid tuples. Let $s \in sales$ and $k \in city[cityId]$

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

Table 3: Referentialities (Definition 5) of foreign key *sales.cityId* values in tuples with invalid or undefined references (see Figures 1 and 2) with different RPPs (Definition 1). Let $s \in sales$ and $k \in city[cityId]$.

$REF(s\,[cityId]\,, k)$	LAX	LON	MAD	MEX	ROM
RPP					
Frequency weighted	2/6	0	1/6	1/6	2/6
Full	1	1	1	1	1
Restricted	0	0	0	0	0
Uniform	1/5	1/5	1/5	1/5	1/5
User-defined (Constant)	0	0	0	1	0

Frequency weighted referential:

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

Figure 6 shows the origin of each equation element and the RPP involved.

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.

Aggregates for max() and min() can be defined also under our framework. The FR and RR variants are defined as follows

$$\begin{aligned} &x\_max(R_i.A, r_i\,[K] = k) \\ &\qquad = MAX(\{r_i\,[A] * REF(r_i\,[K]\,, k) \mid r_i \in R_i\}) \end{aligned} \tag{7}$$

$$\begin{aligned} &x\_min(R_i.A, r_i\,[K] = k) \\ &\qquad = MIN(\{r_i\,[A] * REF(r_i\,[K]\,, k) \mid r_i \in R_i\}) \end{aligned} \tag{8}$$

In order to compute the WR and, specifically, the FWR variants, we need

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

tuple s ∈ sales	REF(s[cityId],LAX)	Equation element
< 1,LAX,..., 54,...>	1	1×54
< 2,LAX,..., 64,...>	1	1×64
< 3,MEX,..., 48,...>	0	0×48
< 4,NXX,..., 33,...>	2/6	$2/6 \times 33$
< 5, η ,..., 65,...>	2/6	$2/6 \times 65$
< 6,ROM,..., 53,...>	0	0×53
< 7,ROM,..., 58,...>	0	0×58
< 8,MAD,..., 39,...>	0	0×39

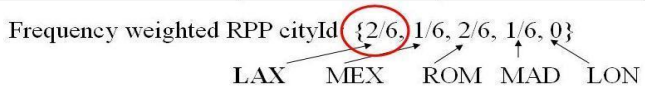

Figure 6: Computation of aggregate fw_sum(sales.amt, s[cityId] = LAX).

to compute the average of the maximum/minimum of the $r_i[A]$ values in R_i of each of the possible ways the defined reference k may be present in the set of tuples with an invalid or undefined reference in foreign key $R_i.K$. Let E_i be this set and e its size. That is, $E_i = \{r_i | r_i \in R_i \ \wedge \ r_i[K] \notin R_j[K]\}$ and $e = |E_i|$. The value that better fits our assumptions as the number of possible ways a given defined foreign key value k may be present in E_i is $\binom{e}{m}$, where $m = \lceil p(k) * e \rceil$, assuming there are tuples with an invalid or undefined reference in $R_i.K$ and $p(k) > 0$. The referentialities of the invalid or undefined references are used here to determine the average a defined reference may be present in E_i. To determine the ways the $r_i[A]$ values in E_i may be present in this average and obtain the average of the maximum/minimum of $r_i[A]$ of all the R_i instances, for a certain defined value k, we procede as follows. Let (a_e) be the sequence of all the $r_i[A]$ values in E_i and, in the case of the w_max() functions, the items of (a_e) ordered in descending order. If $\{r_i \mid r_i \in R_i \ \wedge \ r_i[K] = k\} \neq \phi$, in this sequence, the $r_i[A]$ values that meet the condition $r_i[A] < amax$, where $amax = MAX(\{r_i[A] \mid r_i \in R_i \ \wedge \ r_i[K] = k\})$, are then substituted by $amax$. Due to these substitutions, we may have a different sequence for each k value. Notice also that these sequences may have repeated values.

With these definitions we have

$$w\_max(R_i.A, r_i[K] = k) = \begin{cases} MAX(\{r_i[A] \mid r_i \in R_i\}) & if \ e = 0 \\ \frac{\sum_{i=1}^{e-m+1} a_i * \binom{e-i}{m-1}}{\binom{e}{m}} & otherwise \end{cases} \quad (9)$$

Intuitively, (a_e) has the maximum values of $r_i[A]$ of the r_i tuples with a

defined reference k in $r_i[K]$ of all the $\binom{e}{m}$ instances of R_i where the defined reference k may be present in E_i, ordered in descending order. Thus defined, a_1 will be the maximum of $r_i[A]$ in $\binom{e-1}{m-1}$ instances, a_2 in $\binom{e-2}{m-1}$, and so on, up to complete the $\binom{e}{m}$ instances.

The w_min() functions are defined accordingly, taking sequence (a_e) in ascending order and substituting $amin = MIN(\{r_i[A] \mid r_i \in R_i \wedge r_i[K] \in R_j[K]\})$ instead of $amax$ when $r_i[A] > amin$.

Example 5 *Consider the relaxed database of Figure 1 or the strict database of Figure 2. Let $s \in sales$. To compute $fw\_max(sales.amt, s[cityId] = LAX)$ we procede as follows. The number of tuples with an invalid or undefined value in foreign key $sales.cityId$ is $e = 2$. The number of tuples that best fit the average of tuples value LAX may be present in these tuples is $\lceil p(k) * e \rceil = \lceil (2/6) * 2 \rceil = 1$ considering the frequency weighted RPP. The $s[amt]$ values of the invalid or undefined tuples are 65 and 33, and the maximum value of $s[amt]$ in the valid tuples where $s[cityId] = LAX$ is 64. Then $(a_e) = (65, 64)$. According to Equation 9 we have*

$$fw\_max(sales.amt, s[cityId] = LAX) = \frac{65 * 1 + 64 * 1}{2} = 64.5$$

As for the definitions of the total aggregates, that is, the value of $\mathcal{F}x\_agg()$, using the simplified notation defined in Section 2.2, we have the following:

$$x\_count(R_i.PK) = \sum_{k \in R_j[K]} x\_count(R_i.PK, r_i[K] = k) \tag{10}$$

$$x\_count(R_i.A) = \sum_{k \in R_j[K]} x\_count(R_i.A, r_i[K] = k) \tag{11}$$

$$x\_sum(R_i.A) = \sum_{k \in R_j[K]} x\_sum(R_i.A, r_i[K] = k) \tag{12}$$

$$x\_max(R_i.A) = MAX(\{r_i[A] \mid r_i \in R_i\}) \tag{13}$$

$$x\_min(R_i.A) = MIN(\{r_i[A] \mid r_i \in R_i\}) \tag{14}$$

3.3. Function Properties

Our extended aggregate functions must fulfill certain properties to be considered clean extensions of their counterparts in standard SQL.

Ascending/Descending, An *ascending* feature as defined in [23] holds for the aggregate functions x_count(*), x_count() and x_max(). 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.

Functions x_min() fulfill a *descending* feature. That is, inserting/deleting tuples may imply a decreasing/increasing aggregate respectively.

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

Proof. Since by definition $REF(r_i[K], k) \geq 0$, Definition 5, following the definitions of the extended aggregates x_count(*), x_count() and x_sum() and the FR and RR variants of the x_max() aggregate functions, Equations 4 to 7, we can see that inserting or deleting a tuple in R_i increases or decreases, respectively, the aggregates, no matter if the tuple has a defined, undefined or invalid reference in $r_i[K]$. As for the WR and FWR variants of the x_max(), Equation 9, aggregates, by the definition of the sequence (a_e), in the case a tuple is inserted, the aggregate will only possibly increase when the inserted tuple meets the condition $r_i[A] > amax$, no matter if it has a defined, undefined or invalid reference in $r_i[K]$. Deleting a tuple will only possibly decrease the aggregate since, at most, an item of (a_e) could decrease or could be eliminated. □

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

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

Safety. If our extended aggregations are used in a rigorous database as defined in Section 2.1 a *safety* feature holds 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* $\mathcal{F}x\_agg() = \mathcal{F}agg()$.

Proof. This is easily seen observing that if there are no tuples with an invalid or undefined value in foreign key $R_i.K$, $\forall r_i \in R_i, r_i[K] \in R_j[K]$, then, by Definition 5 we have $REF(r_i[K], k) = 1$, if $r_i[K] = k$ or 0 otherwise, which, in turn means, by the definitions of our extended aggregates, that the tuples that account for a given extended aggregate are the tuples where $r_i[K] = k, k \in R_j[K]$ where, in this context, $R_j[K] = \pi_K(R_j)$ since η is invalid. This is precisely the semantics of the SQL grouped attribute aggregations computed over a joined table on foreign key-primary key attributes [21].
□

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 [25]. That is, a distributive function over a set should preserve the results over the subsets of its partitions. We will prove summarizable consistency for aggregate $\mathsf{w\_count}(R_i.PK)$ ($\mathsf{w\_count(*)}$). The proof is similar for the other WR and FWR aggregates and is based on the definition of referentiality, Definition 5, and the completeness property under an RPP of these type of aggregates, Equation 3.

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

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

Proof. By Definition 4 we have

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

$$= \sum_{k \in R_j[K]} \big(\sum_{\{r_i | r_i[K] \in R_j[K]\}} REF(r_i[K], k) + \sum_{\{r_i | r_i[K] \notin R_j[K]\}} REF(r_i[K], k)\big)$$

$$= \sum_{k \in R_j[K]} \big(\sum_{\{r_i | r_i[K] \in R_j[K]\}} REF(r_i[K], k)\big) + \sum_{k \in R_j[K]} \big(\sum_{\{r_i | r_i[K] \notin R_j[K]\}} REF(r_i[K], k)\big)$$

Using Definition 5 where $r_i[K] \neq k$ and $r_i[K] \in R_j[K]$ we have

$$= \sum_{k \in R_j[K]} \big(\sum_{\{r_i | r_i[K] = k\}} REF(r_i[K], k)\big) + \sum_{k \in R_j[K]} \big(\sum_{\{r_i | r_i[K] \notin R_j[K]\}} REF(r_i[K], k)\big)$$

Using Definition 5 where $r_i[K] = k$ we have

$$= |\{r_i | r_i[K] \in R_j[K]\}| + \sum_{k \in R_j[K]} \big(\sum_{\{r_i | r_i[K] \notin R_j[K]\}} REF(r_i[K], k)\big)$$

Using Definition 5 where $r_i[K] \neq R_j[K]$ we have

$$= |\{r_i | r_i[K] \in R_j[K]\}| + \sum_{k \in R_j[K]} \big(p(k) * |\{r_i | r_i[K] \notin R_j[K]\}|\big)$$

Since the RPP of the WR aggregate functions meets the completeness property we have

$$= |\{r_i | r_i[K] \in R_j[K]\}| + |\{r_i | r_i[K] \notin R_j[K]\}| = |R_i| \tag{15}$$

□

Summarizable consistency for $\mathsf{w\_sum}(R_i.A)$ 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(), sum(), max() and min() total aggregates are invariant with respect to rigorous referential integrity repairs. In our context, a tuple with an invalid or undefined foreign key is rigorously repaired with respect to referential integrity by the substitution of its foreign key value with a defined reference. As the rigorous referential integrity repairs take place, the total aggregate remains *invariant with respect to rigorous referential integrity repairs*. That is, the total aggregate remains constant during this type of repair processes. It is easy to see that, for example, for the w_count(*) aggregate, by Equation 15 in Proposition 4, all the tuples, with or without a defined reference, participate in the total aggregate and since rigorously repairing a tuple, in our context, is equivalent to transfer a tuple from set $\{r_i | r_i \in R_i \wedge r_i[K] \notin R_j[K]\}$ to set $\{r_i | r_i \in R_i \wedge r_i[K] \in R_j[K]\}$, the total remains invariant. A similar reasoning can be applied to the other aggregates. This result is also a consequence of the completeness property (Equation 3) under an RPP of the WR and FWR aggregates. As for the x_max() and x_min() aggregates, it is easily seen from Equations 13 and 14, that the total aggregates do not depend on the value of the foreign key.

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 undefined or invalid references, the defined reference we are considering. For the RR aggregates, this repair consists in never updating the undefined or invalid reference with the defined value we are considering.

3.4. SQL Implementation of Extended Aggregations

In Figure 7 we show a query in SQL over the database in Figure 1, calling sum() and fw_sum() grouped by $city.cityId$, $cityName$, lines 2 to 5, and the equivalent SQL expressions obtaining the same answer set, assuming there exists at least one defined reference in foreign key $cityId$, lines 7 to 26. The corresponding query for database in Figure 2 could be computed with a view of table *city* excluding the tuple marked as undefined.

We first compute a temporal table named fw_temp, lines 7 to 14, with five attributes:

- the values of the attribute $cityName$, renamed as C, from the referenced table that correspond to the primary key $city.cityId$ values referenced by the foreign key $sales.cityId$;
- the foreign key $sales.cityId$ valid values taken from $city.cityId$, renamed as K;
- the number of tuples with a given value in the foreign key $sales.cityId$, $rfreq$;
- the number of rows that have a value different from null in a given attribute, to compute aggregate function count(), is computed from the tuples with a value different from null in $sales.cityId$, $freq$;

```
1: /* SQL query calling extended aggregation */
2: SELECT city.cityId, cityName, sum(amt), fw_sum(amt)
3: FROM city JOIN sales
4:            ON sales.cityId = city.cityId
5: GROUP BY city.cityId, cityName;

6: /* SQL statements evaluating extended aggregation */
7: CREATE TABLE fw_temp AS
8: SELECT city.cityId AS K, cityName AS C,
9:   count(*) AS rfreq,
10:   count(sales.cityId) AS freq,
11:   sum(sales.amt) AS sumagg
12: FROM city RIGHT OUTER JOIN sales
13:   ON sales.cityId = city.cityId
14: GROUP BY K, C;

15: SELECT K as cityId, C AS cityName,sumagg AS sum,
16:        /* sum(R_i.A, r_i[K] = k) */
17: (sumagg + (
18:    /* sum(R_i.A, r_i[K] ∉ R_j[K]) */
19: COALESCE ((SELECT sumagg FROM fw_temp WHERE K IS NULL),0)*
20:        /* p(k) */
21:        (rfreq/
22:          (SELECT sum(rfreq) FROM fw_temp
23:            WHERE K IS NOT NULL )))
24: )  AS fw_sum
25: FROM fw_temp
26: WHERE K IS NOT NULL;
```

Figure 7: Query calling fw_sum() and SQL statements evaluating the extended aggregation.

- finally, the sum() of attribute *sales.amt* for each group, as *sumagg*.

For this implementation we use an alternative way to express fw_sum($R_i.A, r_i[K] = k$). Using Equation 6 and Definition 5 we have:

$$fw\_sum(R_i.A, r_i[K] = k) \\ = sum(R_i.A, r_i[K] = k) + sum(R_i.A, r_i[K] \notin R_j[K]) * p(k)$$

It is easy to see that both fw_count() variants, Equations 4 and 5, have similar alternative expressions. To simplify exposition, η is considered an invalid reference. By computing a RIGHT OUTER JOIN and a GROUP BY in lines 12 and 14 respectively, if there are referential integrity errors, a row with a null value in attributes K and C will be generated holding in attributes $rfreq$ and *sumagg* the number of rows with referential errors and the sum() of attribute *sales.amt* of these rows respectively. In this same row, attribute $freq$ will hold the number of invalid values different from null. Notice that the cardinality of table fw_temp is the number of valid referenced values, plus one in case there are referential integrity errors. The SELECT clause that follows, line 15, computes the sum() and fw_sum() for each value in the element list. We show in lines 16, 18 and 20 the place where each element of the FWR aggregate is computed. In line 17, for each tuple in the answer set, one for each valid reference, *sumagg* holds the sum() of attribute *sales.amt* of all the tuples of the corresponding valid reference. Each one of these values is added to the sum() of attribute *sales.amt* of all the tuples with an invalid reference, or zero if there are no tuples with invalid values (COALESCE clause in line 19), multiplied by the corresponding value in the frequency weighted RPP (dynamically computed in lines 21 to 23). The additional overhead due to the computation of the FWR aggregates comes from the sequential scan of table fw_temp. To compute aggregate functions fw_count(*) or fw_count() we only need to substitute attribute *sumagg* with attributes $rfreq$ or $freq$ respectively in the appropriate places.

We were able to design a clean function invocation for the SQL implementation of the FWR aggregates since the frequency weighted RPP is computed dynamically. For the other WR aggregates, the user is required to provide the corresponding RPP. In Figure 8 we show an SQL implementation using table fw_temp which was computed in the past SQL example and a RPP, $refpp$, with referenced and unreferenced foreign key valid values associated to a probability in the RPP. An additional overhead linear in the number of valid foreign key values should be considered due to the statement in lines 2 to 8. Table $refpp$ has two fields: Kpp holds the referenced and unreferenced values and pp holds their corresponding probability. The user has to be aware that for the WR aggregates, if completeness is to be fulfilled (Equation 3), the probabilities in the corresponding RPP must add to one. The SELECT statement in lines 10 to 16 covers the different cases in order to give a complete, summarizable consistent answer set. In a related work, in [30] we studied the performance of several techniques to compute by means of SQL clauses referential quality metrics. The SQL implementation presented above may be optimized with one of

```
1: /* SQL statement to get unreferenced keys */
2: CREATE TABLE w_temp AS
3: SELECT * FROM
4:   fw_temp FULL OUTER JOIN
5:   (SELECT Kpp, pp, city.cityName AS cc
6:   FROM refpp JOIN city ON city.cityId = refpp.Kpp
7:      WHERE refpp.pp > 0) AS foo
8:   ON fw_temp.K = foo.Kpp;

9: /* SQL statement to compute weighted referential aggregate */
10: SELECT COALESCE(K,Kpp) AS cityId, COALESCE(C,CC) AS cityName,
11:   (COALESCE (sumagg,0) +
12: (COALESCE ((SELECT sumagg FROM w_temp
13:     WHERE K IS NULL AND Kpp IS NULL),0)*
14:   (COALESCE(pp,0)))) AS wsum
15: FROM w_temp
16: WHERE K IS NOT NULL OR Kpp IS NOT NULL;
```

Figure 8: SQL implementation of w_sum() with a RPP in table refpp.

the techniques studied, namely the *early foreign key grouping* technique. This technique evaluates a group-by clause by foreign keys before executing, in our context, the RIGHT OUTER JOIN. The rationale behind this optimization is in reducing the size of the referencing table before joining with the referenced table. In Figure 7, to implement the mentioned technique, the code in lines 8 to 14, may be changed to the one presented in Figure 9. In Section 4 we present several experiments concerning this optimization.

To compute aggregate function w_max() we need to compute the maximum of each group of tuples of each of the defined references in the referencing table, determine sequence (a_e) for each defined value and with the probability of each value and the number of tuples with an undefined reference, according to Definition 9, compute each maximum. Function w_min() is computed accordingly.

3.5. Extended Aggregates Computed on Tables with Columns not Independent

We can improve the estimated answer sets of our extended aggregations computed over a joined table on foreign key-primary key attributes with potential referential integrity violations or with undefined references, if we have another foreign key or other attributes with values of higher quality in the same table (e.g. a foreign key with less referential errors or undefined references). Consider a table with foreign keys belonging to a set of attributes that are not mutually independent. This scenario is possible when two or more databases are integrated and there are tables that share a common primary key. Functional dependencies may be defined between sets of attributes involving foreign keys. Although the database may not be in 3NF, remember we are assuming a relaxed

```
SELECT city.cityId AS K, cityName AS C,
sum(rfreq) AS rfreq, sum(freq) AS freq,
sum(sumagg) AS sumagg
FROM city RIGHT OUTER JOIN
( SELECT cityId AS K,
   count(*) AS rfreq, count(sales.cityId) AS freq,
   sum(sales.amt) as sumagg
 FROM sales
 GROUP BY cityId ) as foo
ON foo.K = city.cityId
GROUP BY cityName, city.cityId;
```

Figure 9: Implementation of early foreign key grouping technique

database or a strict database with undefined references, 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 $R_{j_b}.K_b$, where R_{j_a} and R_{j_b} are two referenced tables, and an attribute $R_i.A$ over which an aggregate function is computed. In our running example *city* may stand for table R_{j_a} and *region* for R_{j_b}, the corresponding foreign keys are $sales.cityId$ and $sales.regionId$ respectively. 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. Now, both foreign keys, $R_i.K_a$ and $R_i.K_b$, may each have a RPP associated, say RPP_a and RPP_b respectively. Let $r_i \in R_i$ where $r_i[K_b] = k_b$ and $k_b \in R_{j_b}[K_b]$. Suppose $r_i[K_a] \notin R_{j_a}[K_a]$, that is, $r_i[K_a]$ is an invalid or an undefined reference. Since $R_i.K_a \rightarrow R_i.K_b$, the subset of defined references that may occur in $r_i[K_a]$, say S_{k_b}, is

$$\{k_a \mid k_a \in R_{j_a}[K_a] \wedge p(k_a|k_b) > 0\}$$

where $p(k_a|k_b)$ is the conditional probability that the defined reference in $r_i[K_a]$ is in fact k_a given that $r_i[K_b] = k_b$. The conditional probability $p(k_a|k_b)$ may not be equal to $p(k_a)$ where $p(k_a) \in RPP_a$.

Based on definition of completeness of a referenced primary key under an RPP, see Definition 1, we say a set of defined references, say S_{k_b}, is complete under the conditional probabilities of its elements given a value, say k_b, if

$$\sum_{k \in S_{k_b}} p(k|k_b) = 1 \tag{16}$$

Consider now the case where $r_i[K_b] \notin R_{j_b}[K_b]$ and $r_i[K_a]$, say k_a, is an element of $R_{j_a}[K_a]$. Then for only one defined value in $R_{j_b}[K_b]$, say k_b, we have $p(k_b|k_a) = 1$ and for the rest, the conditional probability is zero.

Example 6 *Consider the relaxed database of Figure 1 or the strict database of Figure 2. Observe the tuple with value* 5 *in* sales.storeId. *As discussed before, the frequency weighted RPP may be used to compute our extended aggregates. According to Example 3 and assuming η is invalid the frequency weighted RPP is*

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

Now, since the user trusts the foreign key sales.regionId*, the "real" value of the invalid or the undefined reference mentioned above is a city in the* Americas *region. That is*

$$S_{AM} = \{LAX, MEX\}.$$

If we take the referencing relation as $\sigma_{regionId='AM'}(sales)$ the frequency weighted RPP that corresponds to foreign key cityId considering the values of the referenced primary key city.cityId*,* ⟨LAX,MEX,ROM,MAD,LON ⟩*, is*

$$\langle \frac{2}{3}, \frac{1}{3}, 0, 0, 0 \rangle$$

where the probabilities that correspond to LAX and MEX are the following conditional probabilities

$$p(LAX \mid AM) = \frac{2}{3}, \quad p(MEX \mid AM) = \frac{1}{3}$$

considering there are three tuples that correspond to the Americas *region.*

To improve the estimated answer sets of our frequency weighted extended aggregations we can procede as follows. Observe that the set of defined references in $R_i.K_b$ defines a partition of the corresponding set of defined references in $R_i.K_a$. Assuming all the references in $R_i.K_b$ are defined references, we can define a partition of R_i using the values of attribute $R_i.K_b$ and including the tuples with an invalid or an undefined reference in $R_i.K_a$. We can use the tuples of R_i with a defined reference in $R_i.K_a$ to determine the conditional probabilities as in Example 6. Several other cases have to be considered. For the partitions with tuples with defined references in $R_i.K_b$ but having only tuples with invalid or undefined values in $R_i.K_a$, a discreate uniform probability distribution function is defined as in Definition 2 considering only the values in the referenced table R_{j_a} that are not associated to a defined reference in R_{j_b}. In our running example, this could be the case of tuples with invalid or undefined references in *sales.cityId* associated to defined regions that, in turn, are not associated to any defined city. Using a uniform distribution, these tuples will account for part of the corresponding aggregate of the cities that are not associated to any region. Tuples with an invalid or undefined reference in both foreign keys $R_i.K_a$ and $R_i.K_b$ will account for part of the corresponding aggregate of all the defined references in $R_i.K_a$.

As for the tuples with a defined reference in $R_i.K_a$ but with an invalid or

Table 4: Referential aggregate sum(), FWR and FWR improved with trusted foreign key *regionId*. Frequency weighted RPP and conditional probabilities are shown.

cityId	cityName	sum(amt)	fw_sum(amt)	fw_sum(amt) improved	Freq. w. RPP	cond. prob.
LAX	Los Angeles	118	150.6	183.3	0.33	0.66
MAD	Madrid	39	55.3	39.0	0.17	-
MEX	Mexico	48	64.3	80.6	0.17	0.33
ROM	Rome	111	143.6	111.0	0.33	-
-TOTAL	-	316	414.0	414.0	1.0	1.0

undefined reference in $R_i.K_b$, they may be treated as correct tuples by dynamically repairing the tuple with the corresponding value of $R_i.K_b$ obtained from the paired values of the functional dependency assuming there is at least one correct tuple with the defined reference in $R_i.K_b$ for the corresponding defined value in $R_i.K_a$. Otherwise, these tuples are treated as a separated partition.

In case 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 functional 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 by choosing one value.

Take the databases of our running examples in Figure 1 and 2. We show in Table 4 how the aggregation $fw\_sum()$ may be improved by means of the foreign key *regionId*.

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

3.6. 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, $R_j[K]$, the set of referenced defined values, are all the possible defined values. On the other hand, attribute $R_i.K$ is assumed to have potentially invalid references, undefined references and/or the η value, that may be considered valid or not. In this work, the computation of our aggregates is based on the imprecise nature given to the undefined and invalid references. That is, the real defined value of this kind of

references is assumed to be an element of a set, the set of referenced defined values. Although we can design RPPs that fail to meet completeness (see Definition 1), the set of referenced defined values is assumed to be complete. 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 defined value be the actual value that should stand instead of the invalid or undefined value in a foreign key, depends on the occurrence, frequency, of that same defined value in the given foreign key. That is, the occurrence is not completely random, it depends on the observed defined values. On the other hand, we are assuming also that the occurrence of an invalid or undefined value does not depend on the invalid or undefined 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 or undefined 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 defined 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 or undefined reference in attribute K is an independent trial of a given random variable, say the one represented by the potentially defined 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 or undefined reference is updated with value k. That is, it is rigorously repaired with respect to referential integrity with value k.

Notice that if $k \in R_i[K]$ (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 $k \notin R_i[K]$, then the lower bound is 0, that is, no tuples with the defined reference k in $r_i[K]$. In both scenarios, the lower bound represents the case where all the trials (tuples with referential integrity errors or with undefined references) were unsuccessful. That is, the case where the actual value of attribute $R_i.K$ in the tuple with the invalid or undefined foreign key 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 or undefined values were successful. That is, the case where all the actual values of the invalid or undefined values of foreign key $R_i.K$ are indeed k. We can see then that for the random variables described above, if $k \in R_i[K]$ 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 or undefined 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 $k \notin R_i[K]$ 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 (tuples with an invalid or undefined reference) 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, assuming all the tuples with an invalid or undefined reference were rigorously repaired with respect to referential integrity.

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 tuples with an invalid or undefined reference are rigorously repaired with respect to referential integrity. The WR and FWR are the expected value, again, of the corresponding standard aggregates and its result depends on which RPP is considered. This happens to be true also for the extended aggregates x_max() and x_min(), where the RR and FR variants are the lower and upper bound in the case of aggregates x_max() and vice versa in the case of x_min(). As for the WR and FWR variants, assuming $\binom{e}{m}$, (Section 3.2), is the total of all the possible ways a defined reference k may be present in the set of tuples with an invalid or undefined reference, where $e = |\{r_i | r_i \in R_i \wedge r_i[K] \notin R_j[K]\}|$ and $m = \lceil p(k) * e \rceil$, assuming there are tuples with an invalid or undefined reference then, the meaning of Equation 9 in this context is also, as the other WR and FWR aggregates, the expected value of the corresponding standard aggregate.

3.7. Discussion

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

As in Section 3.2, let e be the number of tuples with an invalid or undefined reference in foreign key $R_i.K$, that is, $e = |\{r_i | r_i \in R_i \wedge r_i[K] \notin R_j[K]\}|$. Also, based on the definition of a rigorous referential integrity repair of a tuple with an invalid or undefined foreign key, defined in Section 3.3, let a *rigorous 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 or undefined values of foreign key K replaced with defined 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 running examples, either Figure 1 or 2, there are 25 potential rigorous referential integrity repairs of attribute *sales.cityId* which is a big number considering there are only 2 tuples with invalid or undefined foreign key values. As for the number of plausible values of a given group, assuming a rigorous repair process of a foreign key could 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 examples, 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/undefined 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 3, 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 rigorous 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 4 the corresponding value of the fw_sum() for value LAX, *Los Angeles*, is 150.66. Bear in mind also that our method considers that if there are functional dependencies defined where trusted attributes are involved, like in Example 6 foreign key $regionId$, these functional dependencies may be used to reduce the number of potential referential integrity repairs, refining this way the answer sets of our extended aggregates.

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 rigorous repair process.

4. Experimental Evaluation

We used two real databases and a synthetic database, generated with the TPC-H DBGEN program [34]. For each specific case we report the relational DBMS used. We used standard SQL (ANSI) in our implementation, making it portable in each relational DBMS.

4.1. Real Databases

4.1.1. Government Organization

We present our extended aggregates on a database from a government organization responsible of supervising education services in a state in Mexico (state name omitted due to privacy restrictions). It includes records of 1.7 M enrolled students in 16,000 public and private schools. Evaluation of extended aggregations was carried out on the DBMS Oracle 9i.

The government organization supervises the preschool, elementary and middle school systems. It verifies that certain minimum services are provided such as student evaluations every two months, scholarships, scholastic breakfasts and others.

Every annual cycle each school sends information to a centralized database about its enrolled student population, and every two months sends information

about its active student population. Nevertheless, more than 30% of the registered schools are not connected to the database. These schools represent about 10% of all the student population, but are the schools with the lowest budget. It has been detected that the records that come from these schools have a high incidence of referential integrity errors mostly due to typo errors in the fields of *studentId* and *schoolId*, that is, the foreign keys that reference the tables of the enrolled student population and the registered schools, respectively. Before using extended aggregates, the government organization discarded entirely the tuples with referential errors losing valuable data.

Several assumptions about the database were discussed and were validated by the user in order to obtain valid inferences from the extended aggregates:

- the number of erroneous tuples was proportional to the number of records sent by a given school
- a tuple with an erroneous foreign key in the *schoolId* field came from a school that was not connected to the database
- a referential integrity error did not depend on the particular type of entity that the corresponding tuple came from
- when computing aggregate functions where an attribute is aggregated (e.g. sum()), this attribute did not depend on the foreign key (e.g. the amount received per student did not depend on the *studentId* since all students receive monthly the same amount)

Our extended aggregations have been used to answer queries related to information about how many services and of what type a student or a school have received. By identifying the schools that potentially can send data with referential integrity errors, our method to improve our FWR aggregations, refer to Section 3.5, is being used to obtain better estimations in this set of schools. When the number of referential integrity errors is low, the FR aggregates are used to estimate upper bounds in sums and counts. Results were discussed with the Information Technology manager, a database developer and two users. The IT manager was relief of having an alternative method to respond to the users while he had a solution that had to do with altering the database to correct the referential integrity problems. The database developer was surprise of how quickly this alternative solution was implemented. Users have told us that the computed estimates are useful. Also, they have validated the accuracy of the estimates, since the invalid tuples are fixed during a parallel data cleaning effort. This experience also shows how our techniques can be combined with other strategies.

4.1.2. Retail Company

An important retail company in Mexico listing on the stock market for more than 25 years tried our extended aggregates in one of its applications to assess the usefulness of our approach. Our aggregates were used in a reward program

Table 5: Transaction detail in a given point of sale (POS).

date	store	POS	transId	cli-entId	agentId	prod-uctId	amt	sellerId
02/23/07	213	5	1276	44...47	14545779	14...14	$22,347.83	73...03
02/23/07	213	5	1277	44...47		14...21	$16,086.96	73...03
02/23/07	213	5	1278	15...77	12017425	14...77	$1,477.39	73...03
02/23/07	213	5	1279	21...30	12017378	14...84	$2,826.09	73...03
02/23/07	213	5	1280	21...30		14...25	$773.91	73...03
02/23/07	213	5	1281	43...40		14...18	$513.04	73...03
02/23/07	213	5	1282	23...48		14...84	$4,513.04	73...03

Table 6: 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

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 registers 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 central Oracle database. In this centralized database, about 7% of the total number of sales that should appear with an agent's code, have an invalid value in this field. Table 5 shows several registers of 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 is 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 6 and 7 show the amount of sales per agent, the commission earned and the bonus computed using fw_sum().

4.2. TPC-H Database

Our synthetic databases were generated by the TPC-H DBGEN program [34], with scaling factors 1 and 2. We did not define any referential integrity

Table 7: Agent 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

Table 8: PDFs used to insert invalid values.

PDF	*Probability function*	*Parameters*
Uniform	$\frac{1}{h}$	$h = \|R_j\|$
Zipf	$\frac{1/k^s}{H_{M,s}}$	$M = \|R_j\|$ $s = 1$
Geometric	$(1-p)^{n-1}p$	$p = 1/2$

constraint to allow referential errors. 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. The invalid values were randomly inserted according to three different pdfs, that follow the parameters shown in Table 8, where R_j stands for the referenced table. To evaluate the time performance of the most demanding computation of our extended aggregation, that is, the computation of the auxiliary table *fw_temp* in Figure 7, we used as referencing table *lineitem* and as referenced tables *orders*, *supplier* and *part*. The latter table with 200k registers. The evaluation was done with and without using the optimization technique named early foreign key grouping (see Section 3.4).

4.3. 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 introduced 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

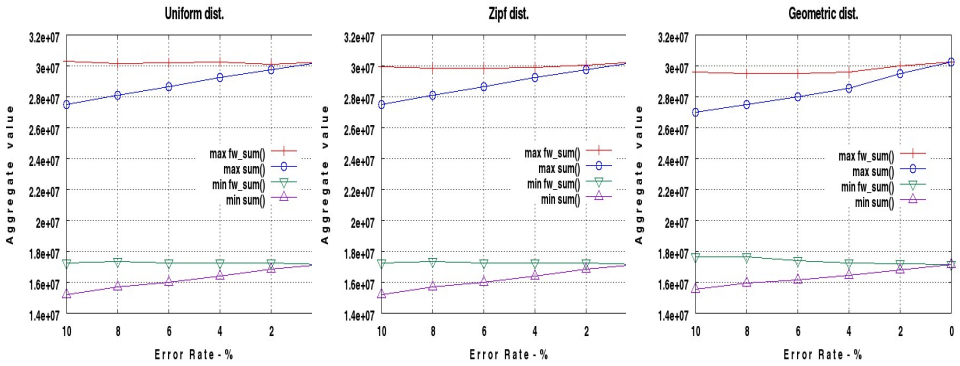

Figure 10: Accuracy of the fw_sum() aggregate function.

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 the answer set. 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 10 shows the accuracy of fw_sum() with attribute *l_extendedprice* in table *lineitem*. The coefficient of variation (σ/μ) of attribute *l_extendedprice* for the invalid tuples was 0.609 meaning that the value of this attribute in the invalid tuples had a low variance. Each plot, one for each pdf, shows 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(). When the error rate is 0% the plotted value on the right side in each plot, corresponds to the totally repaired table. For all the groups with invalid tuples, we computed the average of the absolute difference between the fw_sum() with a rate of 10% of referential integrity errors and the correct sum() and this average was never above 1.46% the value of the average of the corresponding correct element of sum().

In the next experiment we evaluated the approximation accuracy for the WR aggregations, but with different RPP. We simulated the probability that a referential error was in fact a given value following certain pdf. We obtained

Table 9: Value correspondence between w_sum($l\_extendedprice$) computed with different pdfs assuming 10% errors in foreign key $l\_suppkey$ and several statistics. Figures sorted in descending order to show similarities.

PDF	hp/ lp*	w_sum()	$statistic$**	$statistic$ $value$
Constant	hp	22,972,499,088	hp sum() +sum() inv. tup.	22,972,499,088
Geom. ($p = 0.8$)	hp	18,381,997,908		
Zipf. ($M = 10k, s = 1$)	hp	2,366,677,292		
Uniform ($h = 10k$) ***		25,197,285	sum() + avg() inv. tup.	22,361,380
Uniform ($h = 10k$) ***		18,076,153	sum() + avg() inv. tup.	15,204,441
Zipf. ($M = 10k, s = 1$)	lp	15,166,175	lp sum()	15,166,175
Geom. ($p = 0.8$)	lp	15,166,175		
Constant	lp	15,166,175		

* hp - valid reference with highest probability, lp - valid reference with lowest probability
** inv. tup. - invalid tuples
*** For Uniform pdf, valid references that correspond to max(w_sum()) and min(w_sum())

the aggregate values of w_sum($l\_extendedprice$) that corresponded to the valid foreign key value with the highest and lowest probability, hp and lp respectively, with different RPPs assuming 10% errors in foreign key $l\_suppkey$. The *Constant* pdf consists in assigning to one potentially valid reference a probability of one meaning that all the invalid references are, in fact, the corresponding correct reference and, obviously, the rest values have probability zero. Depending on the distribution, the values were different. We computed the sum() and the avg() of attribute $l\_extendedprice$ taking into account only the invalid tuples (i.e. with an invalid reference in attribute $l\_suppkey$; inv. tup. in Table 9), and we also obtained the sum() of attribute $l\_extendedprice$ of the tuples belonging to the valid reference with the highest and lowest probability. In Table 9, we present the different w_sum() values sorted on descending order. We can see a correspondence between the obtained w_sum() values and the statistics. By pairing the aggregate values and the statistics we can see a consistency with the behavior of each of the probability distribution functions. Observe the best estimate for the distribution of the invalid values is the Uniform pdf, which is precisely the pdf used by TPC-H.

4.4. Time Performance

The queries used to compute the WR, FWR and FR aggregations first compute an auxiliary table, fw_temp in Figures 7 and 8. In SQL, this table is computed with a RIGHT OUTER JOIN between the referenced table and the referencing table with several aggregations depending on the function answer set that is needed. For example, for the fw_sum() extended aggregation it computes both count() and sum() for each group and the corresponding values for the invalid references. It also computes the aggregate values of the invalid references, taken such references as a single group. The tuples in this group can

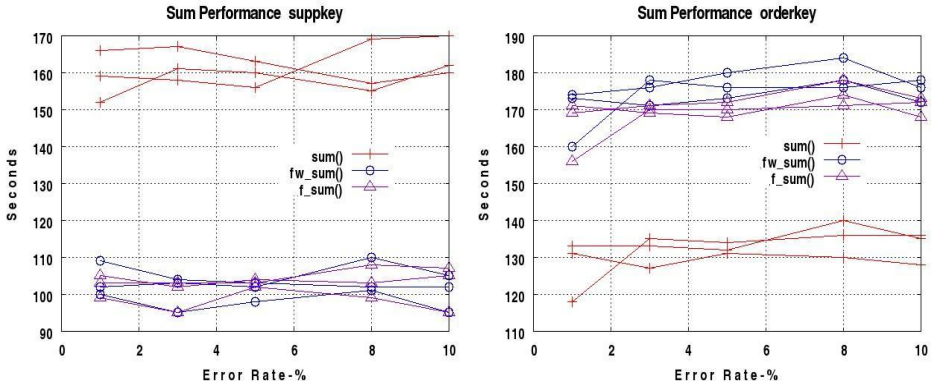

Figure 11: Comparing time performance of aggregations.

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 distinct values that are in the foreign key.

We study the time performance of extended aggregations in Figure 11 for the aggregate functions fw_sum() and f_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 the auxiliary table described earlier. For the sum() aggregations, performance depends on the size of the referenced table, as can be seen in Figure 11. For the WR aggregates, the additional computations are done over the auxiliary table. This overhead is linear in the size of the referenced table.

Since obtaining the auxiliary table prove to be the most demanding computation while computing our extended aggregates, we isolated its calculation and measure its performance in several scenarios. In Table 10 we can see the time it took to compute the auxiliary table with *lineitem* as referencing table, and as referenced tables *supplier*, *part* and *orders*. The computations were measured with and without the early foreign key grouping optimization technique. As we can see, the size of the referenced table plays an important role while measuring time performance.

Summarizing, the performance of our extended aggregations computation depends on the size of the referencing table, the number of invalid values and the

Table 10: Time performance computing auxiliary table with several referenced tables of different sizes and using *lineitem* (6M) as referencing table (time in seconds).

Computation technique	Referenced tables		
	supplier	*part*	*orders*
	10k	200k	1.5M
Early 'group by' optim.	73	83	176
No optimization	239	266	286

number of distinct values in the foreign key attribute. Using the early foreign key grouping optimization technique should be incorporated in an implementation of the extended aggregates.

5. Related Work

Research on managing and querying incomplete data has received significant attention. In [9] 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 [12], 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. Getting consistent answer sets from a query on an isolated database, where some integrity constraints are not satisfied is studied in [8]; 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, where referential integrity constraints are not satisfied. In [7] the authors identify two complementary frameworks to define views over integrated databases and they propose techniques to answer SPJ queries (queries with select, project and/or join operators) where there are missing foreign key values; the authors prove the problem of getting consistent answer sets is significantly difficult (non-polynomial time). In [2] 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. 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 polynomial time. The authors do not address the specific problem of computing aggregations in the presence of invalid foreign keys.

There are several approaches that allow to dynamically obtain consistent answers, that is, answers that do not violate integrity constraints, without modifying the database. In [17] based on query rewriting, the authors proposed a system named ConQuer that retrieves data that is consistent with respect to key constraints given by the user together with their queries. A similar strategy is used in [20], but for consistent answering of conjunctive queries under key and exclusion dependencies. This is done by rewriting the query in Datalog with negation. In [11] the authors present a framework for computing consistent query answers. They consider relational algebra queries without projection and denial constraints. Since their framework can handle union queries, it can extract indefinite disjunctive information from an inconsistent database. All this is done by producing a Java program that computes the consistent answers. In contrast, in our work, an attempt is done to use in some way the inconsistent tuples to obtain an improved answer.

From a data modeling perspective, uncertainty and imprecision have also been handled with extended data models that capture more information about the expected behavior of databases. By defining an imprecise probability data model [27], 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 [27], 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 table. Users can assign these probabilities, but we give a feasible and automated method, exploiting the frequency weighted RPP, to get such probabilities. Reference [4] presents an extended OLAP data model to represent both uncertain and imprecise data. The authors 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. Later, in [5] they enrich these concepts defining extended databases and an extended database model where a probability may be associated to a set of facts where each one of them may represent a possible world. Finally, in [6] the authors extended their previous framework to remove the independence assumption over imprecise facts. We want to stress the fact that this work does not discuss evaluation issues when referential integrity errors occur. Such omission is important because in a database integration scenario, where accurate aggregations are required, tables are likely to have referential integrity errors.

Concerning the properties of the aggregate functions, in [23] the authors define an ascending aggregate function as a monotonic increasing function. Descending functions are defined accordingly. In contrast, in our work, we conceived a new property that has to do with the repairing process of foreign keys. When an invalid foreign key is repaired, that is, when its value is changed by

a valid value, the total value of the aggregate function remains invariant, that is, invariant with respect to referential integrity repairs. Although repairing a foreign key in other contexts may be seen as an insertion of a valid tuple, in our case, since the value of the foreign key is considered as imprecise, the value of the aggregated attribute of the tuple with an invalid foreign key always accounts for an amount of the aggregate total value. Concerning summarizability [25], meaning that a distributive function over a set preserves the results over the subsets of its partitions, since an invalid foreign key value is considered as an imprecise value, summarizability consistency is preserved in almost all cases. A special interesting scenario arises when for all tuples the foreign key holds invalid values. Since a frequency weighted RPP cannot be computed using the frequency of the valid foreign key values, we assume all the referenced values have the same probability.

A closely related research field studies probabilistic databases. Concerning query answering, in [16] the authors present a probabilistic relational algebra where tuples are assigned a probability (weight) of belonging to a relation. The authors define among other operations, the natural join operation for probabilistic relational algebra. In [37] the author uses logic theories based on a probabilistic first order language to formalize probabilistic databases. In [24] the authors assume that the events are not pairwise independent. Using postulates they are able to define classes of strategies for conjunction, disjunction and negation meaningful from the viewpoint of probability theory. Operations such as join must take into account the strategies for combining probabilistic tuples. Also, interval probabilities are considered instead of point probabilities. In our work, we take advantage of functional dependencies to improve our extended aggregates. To use our techniques adequately, the user has to take into consideration the assumptions behind our extended aggregates. On the other hand, these assumptions allow efficiency in the computation of our aggregates. In [13] the authors show that the data complexity of most SQL queries over probabilistic databases is #P-complete. This shows clearly the need of alternatives due to the challenge these type of queries represent.

Specifically, concerning aggregate operators in probabilistic databases in [33] the authors define aggregate operators over probabilistic DBMSs and present linear programming based semantics for computing these aggregate operators. Nevertheless, they prove that in general it is intractable to compute these operators. Also 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. In a recent work done over *Trio* [35], a DBMS for uncertain and probabilistic data, in [28] the authors define aggregations that obtain the answer sets with the lowest, the highest probability and, considering all the values and the probabilities associated to those values, the expected value of an aggregation. The authors bound the aggregations with respect to their probability. In contrast, in our work, we bound our aggregates considering the

value of the answer set. Our lower or upper bounds refer to the lower or upper value an answer set can reach.

To close our discussion on related work, we summarize past research on improving database systems to handle referential integrity issues. In [30] we propose to measure referential integrity errors. In this work, we introduced the early foreign key grouping optimization technique mentioned in Section 3.4, used to obtain the auxiliary table to compute our extended aggregates. Since extended joins involving foreign keys are needed to compute our aggregates, we adapted the optimization techniques so it could be used in our computations. In [31] we generalize referential integrity metrics to distributed databases. We also consider an additional metric to measure consistency in table replicas. In [18] and [29] we presented our initial studies of how to improve aggregations.

6. Conclusions

This work improved SQL aggregations to return enhanced answers sets in the presence of referential integrity errors. Referential integrity errors are manipulated as imprecise values that stand for precise values, determined by a foreign key. We introduced the following families of extended aggregate functions: weighted referential (WR), frequency weighted referential (FWR), full referential (FR) and restricted referential (RR) aggregations. The definition of these extended aggregate functions is based on a new concept called referentiality. Intuitively, referentiality is the degree to which a foreign key value in a referencing tuple refers to some correct key in the referenced table. WR aggregations are based on referential partial probability vectors (RPPs) associated with the foreign key. A particular family of WR aggregations is the frequency weighted referential aggregations (FWR) whose RPP is based on a dynamically evaluated RPP computed from the frequency of tuples with a given reference in the referencing table. FR aggregations represent an extreme repair scenario where each aggregated group receives all the values corresponding to existing referential integrity errors or undefined foreign key values. FR aggregations are helpful when the user needs to include, for each group, all tuples with invalid references. We studied how to compute our extended aggregations considering independent and dependent columns. Our extended aggregations exhibit important properties, which are essential to consider them correct extensions of standard SQL aggregations. A WR aggregation for row counts is summarizable consistent, ascending and safe. Safe means that if no referential errors or undefined values exist, then the extended aggregation result is equal to the result returned by it standard SQL counterpart. A WR sum aggregation is safe and summarizable consistent and when it behaves as an increasing or decreasing function, then it is ascending or descending, respectively. All extended aggregations, together with WR and FR maximum and minimum aggregates, their total aggregate share the invariant with respect to referential integrity repairs property. The maximum extended aggregates are ascending, whereas minimum aggregates are descending; both fulfill the safe property. On the other hand,

FR aggregations are safe and plausible. Plausible means the answer set represents a potential repair for each group, that consists of assigning to all invalid references, the reference that represents each group with a valid key. Our extended aggregations are evaluated with plain SQL queries without modifying the DBMS code, which provides fast and wide applicability. Experiments with two real databases and a synthetic database evaluated usefulness, accuracy and performance.

There are several research issues for future work. Some of our ideas can be extended to more general SPJ queries, especially involving multiway joins. We would like to study the alternative scenario where the referenced table is assumed to be incomplete. Due to the dynamic nature of extended aggregates, we need to improve them with online aggregation techniques for interactive use. We would like to propose a system that based on query rewriting, the user could retrieve, out of standard SQL aggregate queries where foreign keys are involved, alternative SQL queries to obtain consistent answer sets.

Acknowledgments. The first author was sponsored by the UNAM information technology project "Macroproyecto de Tecnologías para la Universidad de la Información y la Computación". We would like to thank the participants of the workshop ACM DOLAP 08 for their interesting comments and feedback and all the anonymous reviewers for insightful comments which helped us to extend an earlier version of this paper.

References

[1] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *ACM PODS*, pages 68–79, 1999.

[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] O. Arieli, M. Denecker, B.V. Nuffelen, and M. Bruynooghe. Database repair by signed formulae. In *FoIKS 2004, LNCS*, volume 2942, pages 14–30. Springer, 2004.

[4] 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.

[5] D. Burdick, P.M. Deshpande, T.S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP over uncertain and imprecise data. *The VLDB Journal*, 16(1):123–144, 2007.

[6] D. Burdick, A. Doan, R. Ramakrishnan, and S. Vaithyanathan. OLAP over imprecise data with domain constraints. *VLDB Conference*, pages 39–50, 2007.

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

[8] 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.

[9] 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.

[10] R. Cheng, D.V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *ACM SIGMOD Conference*, pages 551–562, 2003.

[11] J. Chomicki, J. Marcinkowski, and S. Staworko. Computing consistent query answers using conflict hypergraphs. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 417–426. ACM, 2004.

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

[13] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB Conference*, pages 864–875, 2004.

[14] 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.

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

[16] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[17] A. Fuxman, E. Fazli, and R.J. Miller. Conquer: efficient management of inconsistent databases. In *ACM SIGMOD Conference*, pages 155–166, 2005.

[18] J. García-García and C. Ordonez. Estimating and bounding aggregations in databases with referential integrity errors. In *ACM DOLAP Workshop*, pages 49–56, 2008.

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

[20] L. Grieco, D. Lembo, R. Rosati, and M. Ruzzi. Consistent query answering under key and exclusion dependencies: algorithms and experiments. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 792–799. ACM, 2005.

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

[22] R. Kimball and J. Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons, 2004.

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

[24] L.V.S. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

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

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

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

[28] R. Murthy and J. Widom. Making aggregation work in uncertain and probabilistic databases. In *Workshop on Management of Uncertain Data, VLDB Conference*, pages 76–90, 2007.

[29] C. Ordonez and J. García-García. Consistent aggregations in databases with referential integrity errors. In *ACM International Workshop on Information Quality in Information Systems, IQIS*, pages 80–89, 2006.

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

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

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

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

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

[35] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

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

[37] E. Zimányi. Query evaluation in probabilistic relational databases. In *Selected papers from the international workshop on Uncertainty in databases and deductive systems*, pages 179–219. Elsevier Science Publishers B. V., 1997.