UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

MEASUREMENT AND REPAIR OF REFERENTIAL INTEGRITY ERRORS

T E S I S

QUE PARA OBTENER EL GRADO DE:

DOCTOR EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:

JAVIER GARCÍA GARCÍA

DIRECTOR DE TESIS: DRA. HANNA OKTABA
COTUTOR: DR. CARLOS ORDOÑEZ MONDRAGÓN

México, D.F. 2008.

Dedicada a
Norma, Claudia y Andrea

Agradecimientos

Primeramente deseo agradecer el apoyo invaluable que me otorgó la Universidad Nacional Autónoma de México para la realización de mi trabajo de investigación que culminó en esta disertación. Estoy convencido de que mientras en México existan instituciones como la Universidad, nuestro país siempre tendrá un futuro mejor.

A continuación deseo agradecer el apoyo incondicional que recibí por parte de mis tutores la Dra. Hanna Oktaba y el Dr. Carlos Ordoñez Mondragón. Su guía, sus enseñanzas y sus siempre atinados comentarios fueron un faro que me guió en todo este trayecto. El saber que siempre contaba con ellos me dio conanza y ambos fueron un factor importante para que culminara estos trabajos. Asimismo, el Dr. Sergio Rajsbaum Gorodezky, miembro de mi comité tutoral, fue un aliado que siempre me brindó desinteresadamente sus consejos y opiniones. Su experiencia y visión me ayudaron en todo momento en la realización de los trabajos de investigación. Deseo agradecer también al Dr. Humberto Carrillo Calvet y al Dr. Christopher Rhodes Stephens Stevens, miembros de mi jurado doctoral. Sus comentarios y valiosas opiniones enriquecieron en gran medida los trabajos realizados. Al Dr. Christian Lemaître y León y al Dr. Fernando Gamboa Rodríguez les agradezco el haber revisado y comentado versiones preliminares de mi trabajo de investigación. A todos ustedes amigos, muchas gracias.

Deseo agradecer el apoyo que me brindó el Dr. Ramón Peralta y Fabi, Director de la Facultad de Ciencias por haberme permitido llevar a cabo los trabajos del programa doctoral. Su paciencia, compañerismo, y su contagioso ímpetu, fueron siempre una motivación para seguir adelante. Agradezco al Coordinador del Posgrado en Ciencia e Ingeriería de la Computación, el Dr. Boris Escalante Ramírez y a todo su valioso equipo de trabajo por su disposición para ayudarme siempre en todo lo que requerí.

Agradezco el apoyo recibido por parte del Macroproyecto: Tecnologías para la Universidad de la Información y la Computación, que hizo posible el proyecto del cual formaron parte los trabajos de investigación que aquí se presentan.

Un lugar muy especial ocupan tres personas que fueron como un motor que siempre me animó a continuar con este proyecto y a disfrutarlo plenamente. Norma, Claudia y Andrea, con mucho amor les maniesto mi agradecimiento.

Javier García García
Agosto 2008

Related Publications

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

[55] C. Ordonez, J. García-García, Z. Chen. Measuring referential integrity in distributed databases. In ACM First Workshop on CyberInfrastructure: Information Management in eScience, CIMS, pages 61-66, 2007

[53] C. Ordonez, 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

Abstract

Referential integrity is an essential global constraint in a relational database, that maintains it in a consistent state. This dissertation studies measurement and repair of referential integrity errors. We revisit this classical database problem considering modern applications.

We propose to measure referential integrity errors on centralized and distributed relational databases. We define a set of local and global quality metrics at four granularity levels: database, relation, attribute and value, that measure referential completeness and consistency. Quality metrics are efficiently computed with standard SQL queries, that incorporate several query optimizations. We base the calculations of our metrics on the computation of frequency tables of the foreign key values present in the referencing tables. In a distributed scenario, we present a computation strategy using set reconciliation techniques.

We study the problem where a query in a database involve tables and columns with referential integrity errors. In a query involving SQL aggregations computed over a joined relation on foreign key-primary key attributes, with potential referential integrity violations, tuples with invalid foreign key values are skipped effectively discarding potentially valuable information. This problem is common in an integrated database. We extend aggregate functions computed over tables with referential integrity errors to return complete answer sets in the sense that no tuple is excluded. While computing the aggregation, we dynamically associate to each valid reference, a probability which is the degree to which an invalid reference should actually refer to the valid value. In certain contexts, it is possible to use tuples with invalid references by taking into account the probability that an invalid reference actually be a certain correct reference. This way, improved answer sets are obtained from aggregate queries in settings where a database violates referential integrity constraints. We prove fundamental properties of our extended aggregate functions.

We present an extensive experimental evaluation with real and synthetic databases. We discuss applications of our proposals in several scenarios. Our proposals are useful in database integration, multiple database interoperability and data quality assurance.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Referential integrity is a fundamental global constraint in a relational database [20], that basically ensures a foreign key value exists in the referenced relation. Referential integrity issues are found in database integration, data quality assurance, data warehousing and data modeling. Referential integrity is violated or relaxed for practical reasons. Database integration represents a common scenario where referential problems generally arise because 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 [53]. Therefore, source databases may violate referential integrity and their integration may uncover additional referential integrity problems. Performance is a common reason, where referential integrity checking is disabled to allow fast insertions in batch mode. 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.

The issues outlined above motivated us to revisit the fundamental concept of referential integrity. In short, referential integrity is an important broad problem in modern relational databases. To our knowledge, data quality metrics have not been proposed with respect to referential integrity. Also, the problem of dynamically improve answer sets of aggregation functions has not been studied in the presence of referential integrity violations.

1.1 Thesis Contributions

Referential Integrity Quality Metrics. We propose several Quality Metrics (QMs) that measure completeness and consistency with respect to referential integrity . Our

QMs not only cover normalized databases whose relations are incomplete when they have missing foreign key values, but also measure the inconsistency that arises when a relation is not normalized and an attribute value, determined by a foreign key, does not match the corresponding attribute value in the referenced relation. This is common when tables are denormalized, views are materialized or similar tables, from different source databases, are integrated. Our QMs can be used to measure how well referential integrity has been enforced, or alternatively, how severe referential integrity violations may exist at different granularity levels.

Distributed Referential Quality Metrics. Nowadays many organizations use multiple databases, residing at different locations, that are communicated through the Internet. With the growing usage of the Internet and faster computers, an increasing number of organizations are interested in collaborating with other organizations working at different locations and storing shared data in their local computers. We propose distributed referential quality metrics to efficiently identify tables and columns with referential integrity problems in distributed databases so that users detect and avoid inconsistency or incompleteness issues. In our approach there is the underlying assumption that it is expensive to maintain all databases synchronized with the latest updates. We believe distributed databases combined with referential integrity verification represent a promising alternative to detect and fix data quality issues in order to maintain data repositories in a collaborative environment.

Estimating and Bounding Aggregations in Databases with Referential Integrity Errors. 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 [32, 4] consistency is achieved by inserting or deleting tuples, whereas in [68] 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 [24] and [60]. 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 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 our work we propose a different and innovative strategy in order to obtain improved answer sets produced by queries posed over tables with referential integrity errors that involve aggregation functions. Instead of repairing (changing) the invalid values of a foreign key in the original database, we estimate and bound aggregation answer

sets by using the most likely values from the correct references, this way, performing a dynamic repair. We introduce two separate families of aggregate functions computed over databases with referential integrity errors, the *weighted referential aggregates* and the *full referential aggregates*. The objective of the first family of aggregate functions is to obtain an expected answer set. 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, this way repairing specific values. Our interest is to dynamically and efficiently determine the expected correct values of aggregations where foreign keys are involved. The *full referential aggregates*, on the other hand, give upper or lower bounds of each element of the aggregate list, simulating a dynamic repair where the answer sets represent for each element in the aggregate list, a potential repair which consisted in updating each invalid reference with the value of the correct reference that represents each group. We present a probabilistic interpretation of the extended aggregate functions and show that both families together with the standard SQL grouped attribute aggregations computed over a joined relation on foreign key-primary key attributes with potential referential integrity violations are part of a common probabilistic framework.

SQL to compute QMs and extended aggregates. We introduce a basic set of SQL queries to compute QMs and identify two main optimizations. We also show which query variants are fast, but still reasonably accurate for real data. Our extended aggregate functions are clean extensions of their counterpart standard SQL aggregations. We compare accuracy and speed of query variants on synthetic databases with large relations, where attributes have several basic probability distributions.

1.2 Organization

The chapters of this dissertation are organized as follows:

Chapter 2 Referential Integrity QMs in a Centralized Database - We assume the database may violate referential integrity and relations may be denormalized. We propose a set of quality metrics, defined at four granularity levels: database, relation, attribute and value, that measure referential completeness and consistency. Quality metrics are efficiently computed with standard SQL queries, that incorporate two query optimizations: left outer joins on foreign keys and early foreign key grouping.

Chapter 3 Referential Integrity QMs in a Distributed Database - Distributed relational databases are used by different organizations located at multiple sites that work together on common projects. We focus on distributed relational databases with incomplete and inconsistent content. We propose to measure referential integrity errors in them for integration and interoperability purposes. We propose global referential integrity metrics at three levels: database, relation and attribute. We assume each table can be asynchronously updated at any site and new records are periodically broadcasted to all sites. We explain several distributed query optimization issues. Our

proposal is useful in database integration, multiple database interoperability and data quality assurance.

Chapter 4 Estimating and Bounding Aggregations - Database integration builds on tables coming from multiple databases by creating a single view of all these data. Each database has different tables, columns with similar content across databases and different referential integrity constraints. Thus, a query in an integrated database is likely to involve tables and columns with referential integrity errors. When two tables are joined, normally tuples with invalid foreign key values are skipped effectively discarding potentially valuable information. With that motivation in mind, we extend aggregate functions computed over tables with referential integrity errors on OLAP databases to return complete answer sets in the sense that no tuple is excluded. While computing the aggregation, we dynamically associate to each valid reference, a probability which is the degree to which an invalid reference should actually refer to the valid value. The main idea of this part of our work is that in certain contexts, it is possible to use tuples with invalid references by taking into account the probability that an invalid reference actually be a certain correct reference. This way, improved answer sets are obtained from aggregate queries in settings where a database violates referential integrity constraints.

Chapter 5 Experimental Evaluation - We present an extensive experimental evaluation with real and synthetic databases. We evaluate our local metrics and our extended aggregates, and propose SQL query optimizations, showing they can help in detecting, explaining and repairing referential errors.

Chapter 6 Related Work - This Chapter presents a literature survey of works related to the topics covered in this dissertation. First, the related work for referential QMs is discussed. It is followed by an overview of works related to extended aggregations.

Chapter 7 Conclusions and Future Work - We conclude the dissertation summarizing the insights gained from our studies on the measurement and repair of referential integrity errors in relational databases, and discuss issues for future work.

1.3 Definitions

The following definitions and notations will be used throughout the present dissertation.

Relational Databases

A relational database is denoted by $D(\mathcal{R}, I)$, where $\mathcal{R}$ is a set of N relations (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 relation R_i of degree d_i is denoted by $R_i(A_{i1}, A_{i2}, \ldots, A_{id_i})$, where each attribute comes from some domain. One attribute from R_i is the primary key (PK), called K_i, and the remaining attributes A_{ij} are functionally dependent on K_i: denoted $K_i \rightarrow A_{ij}$. To simplify exposition we use simple PKs. Relations are manipulated with the standard

relational algebra operators $\sigma, \Pi, \bowtie$ (i.e. with SPJ queries [16]) and aggregations. The cardinality (size) of R_i is denoted by $|R_i|$ and n_i (to avoid confusion with N).

Referential Integrity

A referential integrity constraint, belonging to I, between two relations R_i and R_j is a statement of the form: $R_i(K) \rightarrow R_j(K)$, where R_i is the referencing relation, R_j is the referenced relation, K is a *foreign key* (FK) in R_i and K is the primary key (PK) or a candidate key of R_j. In general, we refer to K as the primary key of R_j. To simplify exposition we assume the common attribute K has the same name on both relations R_i and R_j. 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 relation ($R_j.K$ in this case) is not allowed to have nulls. But in general, for practical reasons the foreign key $R_i.K$ is allowed to have nulls when its value is not available at the time of insertion or when tuples from the referenced relation are deleted and foreign keys are nullified [25]. We also study integrity over inclusion dependency (ID). An ID constraint between attributes A_i and A_j from relations R_i and R_j holds when $\Pi_{A_i}(R_i) \subseteq \Pi_{A_j}(R_j)$.

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 $\mathcal{R}$. We refer to the valid state defined above as a *strict state*. A database state where there exist referential errors is called *relaxed state*. In a relaxed database R_i may contain tuples having $R_i.K$ values that do not exist in $R_j.K$.

We add one practical aspect: relations can be denormalized. We compute referential integrity metrics on attributes such that each attribute is either a *foreign key* (as defined above) or a *foreign attribute* (functionally dependent on some foreign key). We use K to refer to a foreign key and F for a foreign attribute, in a generic manner. Recall we assume primary and foreign keys consist of one attribute to simplify exposition, but in practice a foreign key may consist of two or more attributes (i.e. composite keys). Notice a foreign attribute introduces a potential source of inconsistency. Relation R_i has k_i foreign keys and f_i foreign attributes. Attributes that depend only on the primary key, and not on any foreign key, are assumed to be correct. Therefore, we do not define quality metrics on them.

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 relation 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 our work, since there exist several different definitions for aggregate functions [44] 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 [36]. Our proposal can be applied in any database. However, our examples refer to an OLAP database. In this work, the following two relations 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 relation playing the role of the fact table and R_j represents a referenced relation acting as the dimension table. Attribute K is a foreign key in R_i and the primary key in R_j, A is a measure attribute over which the aggregate function is applied.

We are particularly interested in computing aggregations over a joined relation on foreign key-primary key attributes, with potential referential integrity violations, in this case, over $R_i \bowtie_K R_j$. 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 relation 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, we will denote as $R[K]$ the set of values in $\pi_K(R)$ and may or may not include the null value depending on if it is considered valid or not. When null in the foreign key is considered correct, all the tuples with a null in its foreign key are treated as members of a single group of tuples, the null group. Proper explanations will be given for each case. We denote a generic null value by η.

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

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

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

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

One of the main problems we are trying to solve is the following. An inconsistency may arise due to referential integrity errors when group and total aggregates are computed over joined tables:

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

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 equivalent expression written in SQL as

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

Equivalent SQL expressions are given throughout our work since our proposals were implemented and evaluated in a SQL platform.

Distributed Databases

Our proposals are also based on a distributed database at n sites [56]. Let $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ be a set of n relational databases. To have a uniform framework, all databases have the same tables (i.e. same structure), but potentially different content (i.e. inconsistent or incomplete content).

Each table R_i can be updated at any site since all sites have local copies of R_i (replicas) that are periodically refreshed with sets of new records. Since the database is distributed any site can receive local updates and sets of new records.

CHAPTER 2

Referential Integrity QMs in a Centralized Database

In this Chapter we discuss database operations that may violate referential integrity, [54]. We introduce QMs that measure completeness and consistency in a relational database with absolute and relative error. QMs are defined at different storage levels to help the diagnosis and repair of referential errors. Finally we discuss how to compute our metrics using SQL.

2.1 Operations Violating Referential Integrity

Referential integrity can be violated basically for two reasons: (1) a relation (table) coming from a different source database is integrated into a central database, such as a data warehouse. (2) A database operation at the row level introduces a referential integrity violation when a constraint is disabled or non-existent. This is a common scenario in a data warehouse which stores tables coming from multiple source databases. In general, referential integrity constraints are disabled with two reasons in mind: Adding new data more efficiently and avoiding the elimination of tuples with invalid references. The database operations that can cause a relaxed state are: (a) Inserting or updating a foreign key value $R_i.K$ in R_i, such that the $R_i.K$ is null or it does not exist in $R_j.K$ at the time of insertion or updating (completeness issue); (b) Deleting or updating tuples from R_j, whose primary key $R_j.K$ leaves a set of referencing tuples in R_i without a corresponding matching tuple with the same $R_i.K$ value (completeness issue); (c) Updating or inserting an attribute F in R_j, functionally dependent on $R_j.K$, that is used as a foreign attribute in R_i leaving both attributes with inconsistent values or, conversely, updating F in R_i, but not on R_j (inconsistency issue). This case arises when referencing tables are denormalized, when views are materialized or when query

Table 2.1: Logic for acronyms of metrics.

Letter	Meaning
a	absolute error
r	relative error
com	completeness
con	consistency

results are stored.

There are a variety of DBMS mechanisms to enforce referential integrity [25] in databases that are assumed to be in 3NF. A common mechanism is to forbid insert or update operations on a referencing relation that have tuples with inexistent foreign keys on the referenced relation. Orthogonal referential actions associated with delete or update operations on referenced relations commonly are: restrict, cascade, nullify, set a default value or no action at all. These actions are commonly used to neutralize the possible referential integrity violation. In general, a DBMS does not enforce referential integrity on non-key attributes because databases are assumed to be in 3NF and views are materialized. Triggers [25] can be used to detect referential violations.

2.2 Absolute and Relative Error

Let X represent an attribute (foreign key K or foreign attribute F), a relation or a database. We define the absolute error of object X as $a(X)$, given by its number of incorrect references. Let $t(X)$ be the total count of references in X. The relative error of X is defined as $r(X) = a(X)/t(X)$.

Absolute error is useful at fine granularity levels or when there are few referential violations. In contrast, relative error is useful at coarse granularity levels or when the number of referential violations is large.

2.3 Hierarchical Definition of QMs

We introduce referential integrity QMs at four granularities: (1) database; (2) relation; (3) attribute; (4) value. QMs are intended to give a global picture and a micro-view of the database. Database level QMs constitute the first step towards discovering referential errors. Relation level QM isolate those relations with referential problems. Attribute QMs provide specific information that may be used to explain and fix specific referential integrity errors. QMs are hierarchically defined starting with the attribute QMs and ending with the database QMs.

Attribute and attribute value QMs are defined over foreign keys and foreign attributes. QMs over foreign keys represent a measure of *completeness*. QMs over foreign attributes represent a measure of *consistency*. Therefore, denormalization introduces potential inconsistency. Table 2.1 summarizes the logic to name our metrics.

Attribute level QMs

Given an attribute K that is a foreign key, using the notation introduced in Section 1.3, a referencing tuple in R_i, $r_i \in R_i$, is considered correct if there exists a matching tuple in the referenced relation R_j, $r_j \in R_j$, with the same K value, $r_i[K] = r_j[K]$. For a foreign attribute F in a referencing table that is denormalized, a referencing tuple is considered correct if $r_i[F] = r_j[F]$ for the matching tuples r_i and r_j, that is, $r_i[K] = r_j[K]$. Otherwise, a tuple is considered incorrect. Therefore, a tuple in R_i with null values in K or F is considered incorrect. We even consider the case that a foreign attribute is also a foreign key, where we treat it as a foreign attribute so that inconsistency can be detected. Summarizing, referential integrity for K (a foreign key), requires just an existential check, but referential integrity for F (foreign attribute) also requires an equality test.

We define the attribute level metric on R_i with referential integrity $R_i(K) \rightarrow R_j(K)$ on foreign key K. We start by defining the attribute level absolute error metric, that is defined in terms of unmatched values and nulls. If K is a foreign key in R_i referencing R_j then, using the logic to name our metrics in Table 2.1, we have

$$acom(R_i.K) = |R_i| - |R_i \bowtie_K R_j| \tag{2.1}$$

This QM is a measure of completeness. In this case, null values and unmatched foreign keys contribute to absolute error. Motivated by the fact that sometimes foreign keys are allowed to have nulls, especially in data warehouses, we provide a less restrictive definition, where K is allowed to have nulls. In this case error is computed over a subset of R_i, where K is not null. That is, tuples where K is null are assumed correct.

$$acom(R_i.K) = |\sigma_{\text{notnull}_{(K)}}(R_i)| - |R_i \bowtie_K R_j| \tag{2.2}$$

Let F be a foreign attribute in R_i dependent on a foreign key attribute K and let P be the primary key of R_i where R_i references R_j. That is, we have the following functional dependencies: $R_i.P \rightarrow R_i.K$, $R_i.P \rightarrow R_i.F$ and $R_j.K \rightarrow R_j.F$. But notice $R_i.K \rightarrow R_i.F$ is not necessarily valid because $\mathcal{R}$ can be in a relaxed (inconsistent) state. We define the absolute error for a foreign attribute as:

$$acon(R_i.F) = |R_i| - |\sigma_{R_i.F=R_j.F}(R_i \bowtie_K R_j)|. \tag{2.3}$$

This QM is a measure of consistency. Notice the metric for a foreign attribute is strict in the sense that in order to consider a reference a correct one, not only must its foreign key exist, but also have the same value and such value must be different from

null. A comparison between two values in which either value is null returns null by three-valued logic [20] and therefore it increases $acon(R_i.F)$. Notice that in Equation 2.3 we adopted a conservative strategy in the sense that if the foreign key of the corresponding foreign attribute does not match, then it will be counted as an error, no matter the value of the foreign attribute. If F is also a foreign key then we treat it as a foreign attribute so that inconsistency can be detected. If we only need the number of inconsistent values from F where the foreign key is valid we can compute $acon(R_i.F) - acom(R_i.K)$. In an analogous manner, we provide a less restrictive definition, where K can be null. Notice when F is null it will still be considered incorrect since comparison with any value is undefined.

$$acon(R_i.F) = |\sigma_{\text{notnull}_{(K)}}(R_i)| - |\sigma_{R_i.F=R_j.F}(R_i \bowtie_K R_j)|. \tag{2.4}$$

We can now define the attribute level relative error QM.

$$(a)\ \ rcom(R_i.K) = \frac{acom(R_i.K)}{n_i} \qquad (b)\ \ rcon(R_i.F) = \frac{acon(R_i.F)}{n_i} \tag{2.5}$$

This metric is undefined for empty relations. The QM $rcom(R_i.K)$ over a foreign key K is a measure of completeness and $rcon(R_i.F)$ on a foreign attribute F is a measure of consistency.

Value level QMs

We define a value level QM for one foreign key value $k_p \in R_i.K$ as $acom(R_i.K, k_p) = |\sigma_{K=k_p}(R_i)| - |\sigma_{K=k_p}(R_i \bowtie_K R_j)|$, for each key value k_p, where k_p may be null. This QM provides the frequency of invalid key values, including null. This QM is a measure of completeness. The value QM for each foreign attribute value $f_q \in R_i.F$ is similarly defined counting those tuples that contain values $k_p \in K$ and $f_q \in F$ in R_i and $R_i \bowtie_K R_j$. This QM measures consistency and provides the frequency of each non-matching foreign attribute value. If nulls in K are considered correct then f_q will be considered correct if it is null. When nulls are not considered correct a null value in K will cause f_q to be considered incorrect, regardless of null. Therefore, incompleteness in K leads to inconsistency in F. Relative error for attribute values is defined dividing by $n_i = |R_i|$.

Statistics and Correlations on Attribute QMs

We introduce univariate and bivariate statistics to understand the probability distribution behind referential errors in some attribute and also, to discover interrelationships among two unrelated attributes. We calculate univariate statistics on attribute QMs including the minimum, maximum, mean and standard deviation of absolute error (frequency) of invalid attribute values. Statistics on QMs are useful to explain why errors happen and to develop a database repair plan. As we shall see in Chapter 5, these

statistics can give us specific information on the probability distribution behind invalid values in a relation. For instance, we can know how many tuples there are per invalid value on average. To have an idea about the scatter of frequencies of invalid values we compare their mean and standard deviation. The minimum and maximum frequencies help us detect critical values (outliers). We also introduce an error correlation measure between two attributes on the same relation. In general, the correlation between a foreign key and a functionally dependent foreign attribute is close to 1. Error correlation can reveal interesting facts among two attributes that do not have any functional dependency between them. A correlation between two unrelated foreign keys or two unrelated foreign attributes provides valuable insight to explain why referential errors happen.

Relation level QMs

We define relation QMs for table R_i with k_i foreign keys and f_i foreign attributes. When R_i is normalized $f_i = 0$.

Based on attribute QMs we define independent error aggregations on foreign keys and foreign attributes. The following QM measures the completeness of R_i:

$$acom(R_i) = \sum_{j=1}^{k_i} acom(R_i.K_j) \tag{2.6}$$

This QM measures completeness since it tells us how many foreign key values are not matched in each R_j or are simply null. Similarly, we define a consistency QM for R_i as:

$$acon(R_i) = \sum_{j=1}^{f_i} acon(R_i.F_j) \tag{2.7}$$

We define relative error QMs for completeness and consistency, respectively:

$$rcom(R_i) = \frac{acom(R_i)}{k_i n_i} \tag{2.8}$$

$$rcon(R_i) = \frac{acon(R_i)}{f_i n_i} \tag{2.9}$$

$rcom()$ and $rcon()$ are η (undefined) when $k_i = 0$, $f_i = 0$ or $n_i = 0$. Also, $acon(R_i)$ is η when R_i is normalized. Observe that relation level QMs quantify completeness and consistency in a relation based on all its attributes.

Database level QMs

We define database absolute error QMs, for completeness and consistency respectively.

Table 2.2: Attribute level QMs.

Data Quality dimension	absolute error	relative error
Completeness	$acom(R_i.K)$	$rcom(R_i.K)$
Consistency	$acon(R_i.F)$	$rcon(R_i.F)$

$$acom(\mathcal{R}) = \sum_{i=1}^{N} acom(R_i) \tag{2.10}$$

$$acon(\mathcal{R}) = \sum_{i=1}^{N} acon(R_i) \tag{2.11}$$

Finally, we define global relative error QMs.

$$rcom(\mathcal{R}) = \frac{acom(\mathcal{R})}{\sum_{i=1}^{N} k_i n_i} \tag{2.12}$$

$$rcon(\mathcal{R}) = \frac{acon(\mathcal{R})}{\sum_{i=1}^{N} f_i n_i} \tag{2.13}$$

Our database level QMs exclude relations R_i with $k_i = 0$ (e.g. lookup relations without references). If $k_i = 0$ for some R_i then R_i does not contribute to relative error.

Discussion on QMs

Completeness and consistency are measured separately. In our proposal we do not give different weights to references coming from large or small relations. An incorrect reference in a small relation (e.g. a lookup table) is as important as an incorrect reference in a large relation (e.g. a transaction table). Correlations and further multidimensional statistical models can be used to explain relationships between referential errors in different attributes. QMs can be ranked by the cardinality of the referencing relation so that references in smaller or larger relations carry more or less weight based on user's preferences. Table 2.2 summarizes attribute-level QMs, which can be thought as the atomic metrics from which relation and database QMs are aggregated. Value QMs can be considered a zoomed view of attribute QMs. Figure 2.1 shows the hierarchical relationship among QMs; everything is derived from attribute level QMs.

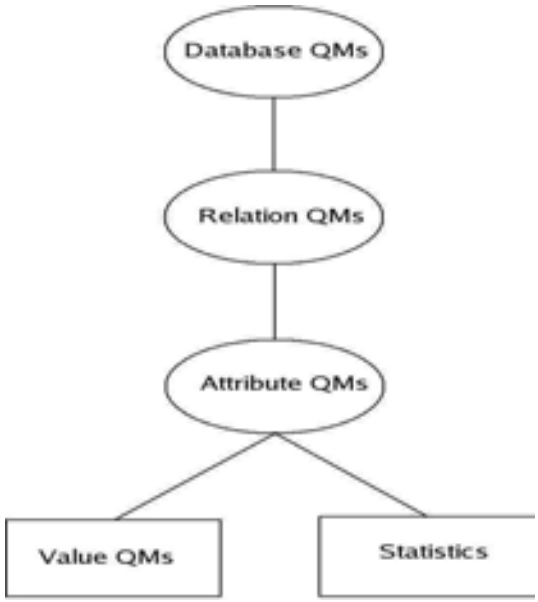

Figure 2.1: QMs diagram.

Table 2.3: Database level QMs

$\mathcal{R}$	N	$rcom(\mathcal{R})$	$rcon(\mathcal{R})$
storeDB	14	0.02%	0.02%

Table 2.4: Relation level QMs

R_i	k_i	f_i	n_i	$acom(R_i)$	$acon(R_i)$
categ	0	0	20	η	η
product	4	5	100109	2444	3100
txHeader	2	2	2003569	0	220
txLine	4	2	19130964	7200	7495

2.4 Example

We now present examples of QMs using a store database schema. Since the goal of our QMs is to help a user discover and explain referential errors in a database, we present QMs going from the highest level down to the most detailed level, in opposite order from their definition. We start showing global measures of quality of referential integrity at the database level in Table 2.3. The first important observation is that $rcom(storeDB)$ and $rcon(storeDB)$ are not zero, which indicates there exist referential errors in the database. The second observation is that incompleteness and inconsistency have similar importance. If it is desired to maintain a database in a strict state this QM table can be periodically refreshed and monitored, by a set of SQL queries, as we will discuss in Section 2.5. Going down one level we can discover which specific relations are giving us problems. Table 2.4 helps us identify those relations with errors. For each relation there are completeness and consistency QMs with $acom()$ and $acon()$. Relation *product* has a mixture of completeness and consistency problems. Relation *txHeader* has only consistency problems in its foreign attributes (e.g. *customerId*). Relation *txLine* reveals completeness and consistency problems: some foreign keys have invalid values or referenced relations are incomplete.

The next level gives us more specific evidence about referential problems, as shown in Table 2.5. Relation *categ* has no references, and it has attribute *categName* which is neither a foreign key or a foreign attribute; therefore, this attribute does not contribute to absolute/relative error. Relation *city* indicates there exists a serious completeness problem in the database with a high fraction of invalid foreign keys. Relation *product* indicates denormalization introduces important consistency problems since it has many inconsistent values in *categName*. In relation *txLine* we can see that the main source of

Table 2.5: Attribute level QMs.

R_i	A_{ij}	*com* or *con*	$a()$	$r()$
categ	categName	$\emptyset$	0	0.00%
city	stateId	*com*	12	9.01%
product	categId	*com*	2444	2.00%
product	categName	*con*	3100	3.00%
txLine	prodId	*com*	7200	0.02%
txLine	prodPrice	*con*	7495	0.02%

Table 2.6: Value level QMs for foreign key K.

$R_i.K$	k_p	$acom(R_i.K, k_p)$
store.stateId	η	4
	0	3
	blank	2
	T.	2
	XX	1

referential problems are invalid foreign keys, but we can see there are 7495-7200=295 rows with inconsistent values for *prodPrice* even when the foreign key exists. Finally, at the finest storage granularity level, Tables 2.6 and 2.7 show value level QMs, exhibiting unmatched foreign key values and inconsistent foreign attribute values in a denormalized relation, as well as their respective frequencies. Values are sorted in descending order by their absolute error to quickly identify critical values in order to develop a database repair plan. In other words, when $acom(K)$ or $acon(F)$ have skewed distributions we can learn which values can help us fix most errors. Value level metrics for F consider

Table 2.7: Value level QMs for foreign attribute F.

$R_i.K$	F	k_p	f_p	$acom(R_i.F, f_p)$
store.cityId	cityName	SF	η	67
		LA	η	43
		SF	blank	34
		SF	Los Angeles	1
		η	New York	1

every existing combination between K and F, which can help explain why denormalized values do not match and assess their relative importance. In this case we can see that the functional dependency between K and F does not hold. When a database is in a *strict* state, these value level tables are empty. Another important aspect is that these QM tables represent extended statistics on the database metadata. Therefore, QM tables can be queried by an end-user to explore a database for completeness and consistency.

2.5 QMs Implementation

Absolute error metrics are distributive [31] based on the fact that they are counts. Therefore, from a query evaluation perspective, most of the effort is spent on computing attribute level or value level QMs. But since relative error is a quotient it cannot be used to compute relative error at coarser granularity levels. This is a consequence of the relative QM being an algebraic [31] function.

We present two query optimizations that are particularly useful to compute QMs. The first optimization favors a left outer join over a set containment computation to compute absolute QMs. The second optimization evaluates error aggregations grouping by foreign keys either before or after joining R_i and referenced relations R_j.

Set containment and left outer join

The following query computes $acom(R_i.K)$ as defined above, with the constraint $R_i(K) \rightarrow R_j(K)$. We call this query "set containment". This query is generally computed with a nested loop algorithm.

SELECT count(*) FROM R_i WHERE $R_i.K_i$ NOT IN
(SELECT $R_i.K_i$ FROM R_i JOIN R_j ON $R_i.K = R_j.K$)

We introduce an equivalent query to compute $acom(R_i.K)$ that may be more efficient, when using a different join algorithm, like a merge-sort join. We call it "left outer join".

SELECT count(*)
FROM R_i LEFT OUTER JOIN R_j ON $R_i.K = R_j.K$
WHERE $R_j.K$ is null

A similar framework can be used for foreign attributes. The following query computes $acom(R_i.K)$ and $acon(R_i.F)$ in a single table scan over R_i. In general, the left outer join optimization will be applied by default.

SELECT
sum(case when $R_j.K$ is null then 1 end) AS *acom*
,sum(case when $R_j.K$ is null or $R_i.F <> R_j.F$ then 1 end)
AS *acon*
FROM R_i LEFT OUTER JOIN R_j ON $R_i.K = R_j.K$;

Early or late foreign key grouping

This optimization evaluates a "group-by" clause by foreign keys either before or after joining R_i and referenced relations R_j. The rationale behind this optimization is reducing the size of R_i before joining with R_j. We also call this optimization the "cube" variant because we build a cube whose dimensions are foreign keys before joining. The early foreign key grouping query optimization with R_i and R_j computes $acom(R_i.K)$ as follows:

INSERT INTO RK SELECT K,count(*) AS *acom*
FROM R_i GROUP BY K;
SELECT sum(*acom*) FROM RK LEFT OUTER JOIN R_j
ON $RK.K = R_j.K$ WHERE $R_j.K$ is null;

On the other hand, the late group-by evaluation on R_i and R_j to get $acom(R_i.K)$ is shown below. The derived table T can be materialized to efficiently query value level QMs.

INSERT INTO Rij SELECT $R_i.K$
FROM R_i LEFT OUTER JOIN R_j
ON $R_i.K = R_j.K$ WHERE $R_j.K$ is null;
SELECT sum(*acom*) FROM (
SELECT K,count(*) AS *acom* FROM Rij GROUP BY K) T;

The early foreign key grouping optimization can be generalized to R_i being joined with m relations $R_1, R_2, \ldots, R_m$ on foreign keys, $K_1, K_2, \ldots, K_m$, respectively.

2.6 Summary

We proposed a comprehensive set of quality metrics (QMs) for referential integrity, which can be applied in data warehousing, database integration and data quality assurance. Our QMs measure completeness in the case of foreign keys and consistency in the case of foreign attributes in denormalized databases. QMs are hierarchically defined at four granularities: database, relation, attribute and attribute value. Quality metrics are of two basic types: absolute and relative error. Absolute error is useful at fine granularity levels or when there are few referential violations. Relative error is adequate at coarser granularity levels or when the number of referential violations is

relatively large. We presented two query optimizations. The first optimization favors a left outer join over a set containment to use a hash or merge-sort join algorithm instead of a nested loop algorithm. The second optimization performs a group-by operation on foreign keys before a join (pushing aggregation, early group-by) to reduce the size of the referencing relation. This optimization is effective for large relations with many foreign keys, where the number of distinct values per foreign key is small.

CHAPTER 3

Referential Integrity QMs in a Distributed Database

In this Chapter we introduce distributed referential integrity metrics, [55]. We explain several distributed query optimization issues. Finally, we conclude by showing how set reconciliation techniques can be adapted to compute our distributed referential integrity QMs. Our proposal generalizes the referential integrity quality metrics for a single database presented in Chapter 2, [54]. We generalized the local metrics to n databases.

3.1 Assumptions

We make two assumptions:

1. Metadata has been integrated before; this is a separate problem.
2. Database content may be inconsistent due to both local and global issues.

We assume table structure is known and uniform across databases. Broadcasting updates happens independently and asynchronously for each table in one database. Therefore, tables may violate referential integrity over time.

3.2 Attribute Level Distributed Metrics

An invalid foreign key or a foreign key with nulls is considered incorrect. In practice nulls are sometimes considered correct FK values. It is straightforward to extend our definitions to ignore nulls for metric computation. We generalize definitions to n databases. Let $\mathcal{T}_\cup$ be defined as the global union of all local copies of table R:

$$\mathcal{T}_{\cup} = D_1.R \cup \cdots \cup D_n.R. \tag{3.1}$$

Observe that identical tuples that appear in different replicas will appear once in the global union. Let $\mathcal{T}_{\cup_i} = D_1.R_i \cup \cdots \cup D_n.R_i$ be the referencing global union and $\mathcal{T}_{\cup_j} = D_1.R_j \cup \cdots \cup D_n.R_j$ the referenced global union. The next metric measures global completeness for $R_i.K$ and detects the existence of table replicas with missing foreign keys in K.

$$grcom(\mathcal{T}_{\cup_i}.K) = \frac{|\mathcal{T}_{\cup_i}| - |\mathcal{T}_{\cup_i} \bowtie_K \mathcal{T}_{\cup_j}|}{|\mathcal{T}_{\cup_i}|} \tag{3.2}$$

Contrast Equation 3.2 with Equation 2.5(a). Observe that if the same erroneous tuple appears in several replicas it will be accounted as one erroneous tuple. This is motivated by the fact that an inserted erroneous tuple in a replica will eventually be replicated to all the local instances. If the tuple was being deleted while running the metric computation, the result will show the pessimistic case. That is, it will count the erroneous tuple.

We now concentrate on consistency metrics. Let F be a foreign attribute in a denormalized table R_i. Recalling Equations 2.3 and 2.5(b), the following equation measures the local consistency:

$$rcon(R_i.F) = \frac{|R_i| - |\sigma_{R_i.F=R_j.F}(R_i \bowtie_K R_j)|}{|R_i|}. \tag{3.3}$$

This equation can be computed in a simpler manner with one relational operator as:

$$rcon(R_i.F) = \frac{|R_i| - |R_i \bowtie_{K,F} R_j|}{|R_i|}. \tag{3.4}$$

We generalize the previous definition to n databases reusing the global union tables $\mathcal{T}_{\cup_i}, \mathcal{T}_{\cup_j}$. The following metric detects foreign columns F that do not match their corresponding reference values in the referenced table.

$$grcon(\mathcal{T}_{\cup_i}.F) = \frac{|\mathcal{T}_{\cup_i}| - |\mathcal{T}_{\cup_i} \bowtie_{K,F} \mathcal{T}_{\cup_j}|}{|\mathcal{T}_{\cup_i}|} \tag{3.5}$$

3.3 Table Metrics

We first define a metric to know if a table has all latest updates (i.e. if it is current). This global metric is applicable to both the referencing table R_i and the referenced table R_j and is similar to the Jaccard coefficient [34]:

$$gcur(R_i) = \frac{|D_1.R_i \cap D_2.R_i \cap \cdots \cap D_n.R_i|}{|D_1.R_i \cup D_2.R_i \cup \cdots \cup D_n.R_i|} = \frac{|\mathcal{T}_{\cap_i}|}{|\mathcal{T}_{\cup_i}|} \tag{3.6}$$

where $\mathcal{T}_{\cap_i}$ is the global intersection.

Given a global union $\mathcal{T}_{\cup_i}$ with k foreign keys $K_1, \ldots, K_k$ its referential completeness is measured by:

$$grcom(\mathcal{T}_{\cup_i}) = \frac{\sum_{j=1}^{k} grcom(\mathcal{T}_{\cup_i}.K_j)}{k} \tag{3.7}$$

Given the global union $\mathcal{T}_{\cup_i}$ with f foreign columns $K_1, \ldots, K_f$ its referential consistency is:

$$grcon(\mathcal{T}_{\cup_i}) = \frac{\sum_{j=1}^{f} grcon(\mathcal{T}_{\cup_i}.F_j)}{f} \tag{3.8}$$

3.4 Database Metrics

Finally, we define global metrics for the entire distributed database $\mathcal{D}$ for completeness and consistency, respectively:

$$grcom(\mathcal{D}) = \frac{\sum_{j=1}^{N} |\mathcal{T}_{\cup_j}| grcom(\mathcal{T}_{\cup_j})}{\sum_{j=1}^{N} |\mathcal{T}_{\cup_j}|} \tag{3.9}$$

$$grcon(\mathcal{D}) = \frac{\sum_{j=1}^{N} |\mathcal{T}_{\cup_j}| grcon(\mathcal{T}_{\cup_j})}{\sum_{j=1}^{N} |\mathcal{T}_{\cup_j}|} \tag{3.10}$$

Contrast Equations 3.9 and 3.10 with Equations 2.12 and 2.13 respectively. Absolute distributed QMs are computed accordingly. The diagram in Figure 3.1 shows our proposed metrics classification according to data quality dimensions, being local or global and absolute or relative, and *gcur* as a separated metric.

3.5 Query Optimizations

Generalizing optimizations to n databases is interesting because updates can happen asynchronously at any site. We adapted distributed query optimization techniques [56] to our problem, assuming the cost to transfer a large table is high. For global metrics the most important relational operation is the global union of all local copies of one table. Such union can be optimized by computing pairwise intersections of tables projecting only the foreign key to determine which tables have new rows and then transfer only new rows. Metrics are based on foreign keys and foreign columns which can be projected before the global union. The second important operation is computing the global intersection to find out records existing at every database. The intersection operation is less demanding because intersections generally return smaller tables.

Global metrics for all tables (say N) and the database can be computed with several approaches:

- A static approach is to transfer $n-1$ copies to some site designated as the "central" site. When this is done for all N tables it becomes expensive.

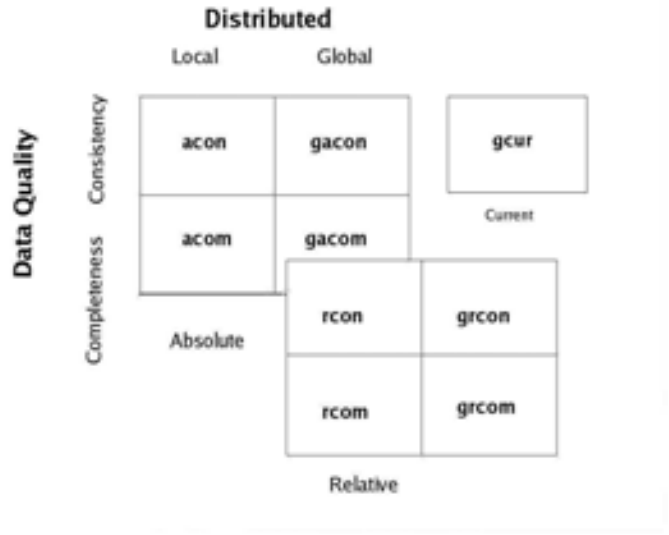

Figure 3.1: Classification of metrics.

- A second approach is to compute metrics at one site one time and then incrementally update them when new records are inserted; metrics are continuously recomputed for all N tables when they are refreshed. The challenge is to maintain a low overhead.
- A third approach is to compute metrics for each pair of tables linked by a foreign key on demand, when their referential integrity needs to be verified.

When tables are large, sampling can be used to get an approximation of referential metrics. Iterated sampling are preferred over transferring large tables. To process a distributed join the program can take advantage of a replicated table, being locally stored at the same site, but which may be outdated. When there is a need to join tables stored at different sites the program projects only a minimum set of primary keys, foreign keys and foreign attributes, covering the generated query. When a join requires two tables stored at different sites the smallest table is transferred to the other site. Set reconciliation algorithms can be adapted to the computation of our distributed referential integrity metrics as we explain next.

3.5.1 Set Reconciliation Techniques

Observe that the computations of our global metrics are based on counts over expressions with the following relational algebra binary operators: intersection $\cap$ union $\cup$ and join $\bowtie$. Also observe that the tables used to compute each global table $\mathcal{T}_{\cup_i}$ and the ones used to compute the metric *gcur* for a given table are replicas (of the same table). We can expect that the number of differences among the replicas of a given table that

participate in a union or intersection operation is low. The number of tuples could be very large, maybe millions of tuples. However, the number of differences among replicas is expected to be low. An alternative to compute the binary operations involved in our metrics could be, as we explained above, to transfer an entire operand from one site to another in order to keep the two operands involved in one site and then compute the operations, for example, computing an antijoin [38]. This alternative could be very expensive in terms of communication complexity since the operands may be tables with millions of tuples. Although a projection of one of the operands could be transfered instead since we are only counting tuples, still the amount of data to be transferred could be too much. The amount of data to be transfer ed may be dramatically reduced considering the following ideas.

As for the computation of global tables observe that

$$\begin{aligned}\mathcal{T}_{\cup} &= D_1.R \cup \cdots \cup D_s.R \cup \cdots \cup D_n.R \\ &= D_s.R \cup (\bigcup_{\substack{k=1 \\ k \neq s}}^{n} (D_k.R - D_s.R))\end{aligned}$$

That is, to compute the union of, for example, two replicas, say $D_a.R$ and $D_b.R$, in a given site, say site a, aside from $D_a.R$ we need the difference $D_b.R - D_a.R$, since

$$D_a.R \cup D_b.R = D_a.R \cup (D_b.R - D_a.R)$$

Likewise, to compute the union in site b we only need $D_a.R - D_b.R$. Note that since $D_a.R$ and $D_b.R$ are replicas (of the same table), the number of tuples that belong to the difference between the two replicas is expected to be low.

Note that since $D_a.R \cap D_b.R = D_a.R - (D_a.R - D_b.R)$ to compute the intersection of replicas we have

$$\begin{aligned}\mathcal{T}_{\cap} &= D_1.R \cap \cdots \cap D_s.R \cap \cdots \cap D_n.R \\ &= D_s.R - (\bigcup_{\substack{k=1 \\ k \neq s}}^{n} (D_s.R - D_k.R))\end{aligned}$$

Again, we only need the corresponding differences in one site to compute the intersection.

As for the join, $\bowtie$, this operator is computed on foreign key-primary key attributes with potential referential integrity violations. In our metrics, this operator is used to count the number of tuples that correspond to the valid references in the referencing table. Stated differently, to achieve the same goal we can compute the set of referencing values that are not in the set of referenced values, that is, the set of invalid references. Afterwards, count the tuples of the referencing table that do not hold these values or do not have a null value in the foreign key, when we assume null is invalid. For example,

we can compute the metric $grcom(\mathcal{T}_{\cup_i}.K)$ as follows. Let $\mathcal{T}_{\cup_i}$ and $\mathcal{T}_{\cup_j}$ be the referencing global union and the referenced global union respectively. Suppose also that site s holds $\mathcal{T}_{\cup_i}$ and in this site we decided to compute $grcom(\mathcal{T}_{\cup_i}.K)$. The metric can be computed as follows

$$\begin{aligned} grcom(\mathcal{T}_{\cup_i}.K) &= \frac{|\mathcal{T}_{\cup_i} \bowtie_K \mathcal{T}_{\cup_j}|}{|\mathcal{T}_{\cup_i}|} \\ &= \frac{|\mathcal{T}_{\cup_i}| - |(\Pi_K(\mathcal{T}_{\cup_i}) - \Pi_K(\mathcal{T}_j)) \bowtie_K \mathcal{T}_i| - |\sigma_{K\ is\ \eta}(\mathcal{T}_{\cup_i})|}{|\mathcal{T}_{\cup_i}|} \end{aligned}$$

To compute the above expression in site s, among other computations we need the difference $\Pi_K(\mathcal{T}_{\cup_i}) - \Pi_K(\mathcal{T}_{\cup_j})$. This expression computes a table with the foreign key invalid values. Observe that if we suppose that the difference is low, this table difference could be small compared to the size of $\mathcal{T}_{\cup_j}$ or even $\Pi_K(\mathcal{T}_{\cup_j})$. Metric *grcon* can be obtained following the same ideas using Equation 3.5.

The question that remains now is how can we efficiently compute the mentioned differences without transferring large amounts of data. In the case of the union and the intersection in our examples, how can we compute in site a the differences $D_a.R_i - D_b.R_i$ or $D_b.R_i - D_a.R_i$ without transferring one of the tables involved to the corresponding site. In the case of the join operation we need to efficiently compute $\Pi_K(\mathcal{T}_{\cup_i}) - \Pi_K(\mathcal{T}_{\cup_j})$ without transferring one of the global tables or the corresponding projection. To accomplish this goal, we propose to adapt techniques used to reconcile sets whose differences are small [48], that avoid the need to transfer large amounts of data. The amount of data transferred depends on the number of differences rather than on the size of the tables involved although once the data is transferred, the computation complexity of the algorithms that compute the actual different values could be cubic in the number of differences.

3.5.2 Global Operations

The techniques we adapted consider the following assumptions and actions:

- The sets to be reconciled are represented by their characteristic polynomials.
- A number of evaluation points mutually agreed must be evaluated using each one of the characteristic polynomials. The number of points evaluated must not be less than the number of symmetrical differences.
- In case that an upper bound of the number of differences can not be determined a priori, there are probabilistic techniques to compute the differences [49].
- Instead of transferring one of the reconciling sets from one site to the other, the evaluation points with their corresponding characteristic polynomial evaluations are transferred, this way minimizing the communication complexity.

- Both sets of characteristic polynomial evaluations are used to interpolate a rational function which will have as roots of its numerator and denominator the symmetrical differences.
- The missing values are obtained by reducing the rational function and finding the roots of the numerator or the denominator.

The following example gives an overview of how the mentioned techniques can be used to reconcile two replicas of a table, although it is not our intention to give here all the technical details of the set reconciliation algorithm (See [49] for details).

Example 3.1. The following example uses the synthetic database generated with *TPC-H DBGEN* program [64]. Suppose table *orders*, with approximately $1.5M$ tuples is replicated at two sites, say site a and site b in an asynchronous multi-master replication scenario. Assume we know the primary key *o_orderkey* holds positive integers stored in a 32-bit signed integer. Suppose the number of symetrical differences between the two sets of primary key values is no more than 50. To reconcile both replicas, first both sites agree in 50 evaluation points, one for each possible different value. At each site the characteristic polynomial, that is, the polynomial whose roots are all the primary key values of the corresponding replica, is evaluated in each one of the 50 evaluation points. Let $k_i, i = 1 \dots 1.5M$ be the primary key values of the replica at site a. The corresponding characteristic polynomial would be then $\Pi_{i=1}^{1.5M}(x - k_i)$. Let k_e be one of the 50 evaluation points. The evaluation of k_e using the mentioned characteristic polynomial is $\Pi_{i=1}^{1.5M}(k_e - k_i)$. To avoid working with very large numbers while computing the evaluations of the characteristic polynomials, we need to work in a finite field with an order not less than $v = (2^{31} - 1) + 50$ in order to map the primary key values and the evaluation points. The selected field could be then $\mathbf{F}_q$, q= 2147483713, being the value of q the lowest prime greater than v. Observe we can fix the value of q since the domain of the primary key values does not change as well as the upper bound of the number of symmetrical differences. The 50 mutually agreed evaluation points may be the integers in the closed interval $[-49, 0]$. Since these numbers are not members of the set of possible primary key values, they will never be zeros of the characteristic polynomials. Both sites keep an updated vector, lets call it the evaluation point vector, with 50 pairs of values containing each evaluation point associated to its corresponding evaluation of the characteristic polynomial. Also, both sites keep the cardinality of their replica. To keep the vector updated, whenever a tuple with a primary key value of say k_0 is inserted/deleted, all the 50 current values of the characteristic polynomial have to be multiplied/divided (mod q) by $(k_e - k_0)$, where k_e is an integer in $[-49, 0]$. The cardinality is updated as well by adding/subtracting one to the current cardinality. Suppose that in a given moment both replicas agree in all the primary key values that cover the positive integers in the closed interval $[1, 1'500, 000]$. In both evaluation point vectors the value that corresponds to the evaluation point -1 is

$$929792600 = (-1 - 1)(-1 - 2)...(-1 - 1'500, 000) \bmod q$$

Suppose site a receives a tuple with value $1'600,000$ in its primary key and site b receives one with value $1'700,000$ in its primary key. The updated values of the characteristic polynomials that correspond to the evaluation point -1 would be 252388150 and 594709911 respectively.

When the set reconciliation is required, both sites exchange their evaluation point vector and the cardinality of its corresponding replica. Observe that the size of the transferred data depends on the number of symmetrical differences and not on the table cardinality. At each site, with both vectors, at each evaluation point, the two values of the evaluation of the characteristic polynomials are divided (mod q) and the divisions are used together with the cardinality of the functions, to interpolate a reduced rational function which will have in its numerator and its denominator the characteristic polynomials of the differences of each set of primary key values. Take for example the values of both vectors that correspond to the evaluation point -1 above. The division between both values gives

$$\begin{aligned}\frac{252388150}{594709911} &= \frac{(-1-1)(-1-2)...(-1-1'500,000)(-1-1'600,000)}{(-1-1)(-1-2)...(-1-1'500,000)(-1-1'700,000)} \\ &= \frac{(-1-1'600,000)}{(-1-1'700,000)} = \frac{2145883712}{2145783712} = 501706218\end{aligned}$$

The above equations are written to highlight the key aspect of this technique. Observe that the common factors cancel out so the result is the evaluation of a rational function whose zeros in the numeration and the denominator are the different values. In case the maximum number of differences of each replica is not known and we only count with a global bound, the cardinality of the tables together with the global bound can be used to determine the bounds of the degrees of both characteristic polynomials in order to interpolate and reduce the rational function. By finding the zeros of the corresponding characteristic polynomial, each site is able to determine the primary keys of the tuples that the other site is missing and can send to its counterpart the missing tuples.

In case the keys are not integers, the key domain values are mapped to the values of a finite field. This can be done considering their binary representation.

Note that to compute our distributed metrics, we do not need the complete set of tuples at all the sites where the replicas reside as in the set reconciliation problem. Also, we focus on counting tuples instead of determining precise different values. Next, we show how to use the set reconciliation techniques mentioned above in order to compute the global operations required to compute our metrics.

Global Union $\mathcal{T}_{\cup}$

Observe that to compute our metrics, the information needed from the global tables is the set of values of the primary keys and the set of values of foreign keys and foreign

columns that are to be measured. We will first assume that the maximum number of symmetric differences among any two sets is less than an upper bound, say d_p. Later on we will study the case where the number of differences cannot be bounded.

To compute the global union, say $\mathcal{T}_{\cup_i}$, we have to keep at each site where a replica of R_i resides, the evaluation of the characteristic polynomial of the primary key of R_i at d_p evaluation points mutually agreed among all the sites that hold a replica of R_i. At each site we will keep a vector with d_p pairs of values each pair containing an evaluation point and its characteristic polynomial evaluation. We will call this vector *evaluation point vector*. Observe that the evaluation points should be chosen so that they are not zeros of the characteristic polynomials to avoid complications in the interpolation of the rational function. Note that the evaluations can easily be updated each time a tuple is inserted or deleted. The cardinality of each replica of R_i is also maintained at each site. To evaluate the global table, a site is chosen to eventually hold the global table. Let s be that site. This site broadcasts its evaluation point vector. Let site $u \neq s$ represent each one of the sites where a replica of R_i resides. With the evaluation point vector of s and its own, site u computes the values of the primary keys of R_i that are at site u and not at site s and transfers to site s the tuples that correspond to the computed missing primary key values. Observe that instead of the complete tuples, site u only needs to transfer the primary key and the foreign key and/or foreign columns that are required depending on the metric that is evaluated. Site s builds a table with the same schema of R_i, say R_i^+ and inserts in it the missing tuples sent by the other sites. Observe that since R_i^+ has the same schema as R_i, then duplicate tuples will be rejected. The global table $\mathcal{T}_{\cup_i}$ will be $R_i \cup R_i^+$ and its cardinality can easily be computed since site s maintains the cardinality of (its replica) R_i and only $|R_i^+|$ has to be computed.

Global Intersection $\mathcal{T}_{\cap}$

The intersection of the replicas of a given table can be obtained in a similar fashion. Take sites s and u as defined in the previous section. Again site s broadcasts its evaluation point vector. Now site u transfers the values of the primary keys that are in s but not in u. Site s builds a table, say R_i^-, with a single column containing the primary key values eliminating duplicates. The size of the intersection is then $|R_i| - |R_i^-|$.

Observe that to compute *gcur* at site s, the computation of the union and the intersection may be done simultaneously and s needs to broadcast only once its evaluation point vector, and the corresponding sites send the required values of R_i^+ and R_i^-.

Example 3.2. Following our running example, suppose now there are several replicas of table *orders*. The site that will hold the global union or the site where the metric *gcur* is to be computed, say site s, broadcasts its evaluation point vector in order that the involved sites where the other replicas of table *orders* reside compute the values of the primary keys of table *orders* that are needed depending on the desired metric. For the global union, they are required to send the primary key values that are in their

replica but not in the one at site s. For the global intersection they are required to send the primary key values that are at site s and not in their replica.

Valid References

To compute the valid references of a referencing global union, once the involved tables are computed as in the global union, see page 27, if both tables are at different sites, we procede the following way. Let r be the site with the referencing global union, say $\mathcal{T}_{\cup_i}$, and s the one with the referenced global union, say $\mathcal{T}_{\cup_j}$. Suppose that the maximum number of differences between $\Pi_K(\mathcal{T}_{\cup_i})$ and $\Pi_K(\mathcal{T}_{\cup_j})$, the set of foreign key values of the referencing global union, and the set of primary key values of the referenced global union, is less than an upper bound, say d_q. The case where an upper bound cannot be established will be treated later on. Suppose $d_q = d_p$, if not, an additional evaluation point vector should be computed for each referenced table at the site of the replica where the global referenced union is computed and we procede using this table instead. At site s we compute a temporal evaluation point vector from the evaluation point vector of the replica R_j in s as if the tuples of table R_j^+ were inserted in R_j. By doing so, we obtain the evaluation point vector of $\Pi_K(\mathcal{T}_{\cup_j})$.

As for the referencing global union $\mathcal{T}_{\cup_i}$ in r, the evaluation point vector for $\Pi_K(\mathcal{T}_{\cup_i})$ has to be computed from scratch. A frequency table is built with the values of the foreign key of the global union $\mathcal{T}_{\cup_i}$ excluding η since we are considering η an invalid foreign key. The idea is to keep the number of tuples for each foreign key value . While doing so, with the d_q evaluation points we compute the corresponding evaluation point vector. Later on, the frequency table will be used to compute the number of valid tuples. Site r transfers the computed evaluation point vector to site s where the values of $\Pi_K(\mathcal{T}_{\cup_i}) - \Pi_K(\mathcal{T}_{\cup_j})$ are computed. The set of values are in turn transferred to r and using the frequency table the number of valid references is obtained.

Communication and Computational Complexity

The communication complexity to compute our metrics using the set reconciliation techniques just described depend linearly on the number of differences among the sets of values involved. Let d be the maximum number of differences between two replicas. To compute the *gcur* metric one of the sites that holds a replica, say site s, has to broadcast only once an evaluation point vector with its size bounded by d. In return, it receives from the sites that have a replica, take site u as one of these sites, the values that belong to the symmetrical difference between the primary key values of s and u. This is done to compute the size of the union and the intersection. Computed this way, the communication using set reconciliation techniques has complexity $O(d)$. At each site that holds a replica, the computation to interpolate the rational function using the evaluation point vectors by Gaussian elimination and the Euclidean algorithm to reduce the rational function, has complexity $O(d^3)$.

To compute *grcom* and *grcon* we need a frequency table with the number of tuples per value for each foreign key to be measured. This table can be computed by several means. Via a simple scan of the global referencing union if a table containing the different values and their frequencies can be stored in memory or sorting a projection of the referencing global union containing its primary key and its foreign key and then computing the needed table. Notice that in case of *grcon* we need the foreign key and the foreign column. In the worst case, the cost is $O(n\ log(n))$ where n is the size of the referencing global union. Observe that while computing the frequency table, the evaluation point vector can be computed simultaneously. Observe that to compute our metrics we can use the frequency table instead of the referencing global union.

Unknown Upper Bound

In many cases an upper bound of the number of symmetrical differences, d, is not known. A more realistic scenario is that an upper bound is not known. We present an overview of a method to apply the techniques proposed without knowledge of an upper bound, considering first two tables, $D_a.R_i$ and $D_b.R_i$. This method is an adaptation of the probabilistic verification presented in [49].

We estimate a maximum number of symmetrical differences above which it would be unfeasible to use this method in contrast to the one that consists in transferring a projection of the table needed depending on the metrics we are computing. Observe that the real number of differences could be bigger. We determine a first set of evaluation points mutually agreed between the involved sites with this number of points. Lets call this set Q. Next, we need to determine a number of additional evaluation points that will be used to test with a given probability if an interpolated rational function is indeed the function we are looking for. To do this, we determine a low probability, p, representing the maximum probability we can tolerate for the case where we erroneously interpolate a rational function different from the correct rational function whose zeros of the numerator and the denominator constitute the values of the symmetrical differences. According to this probability we determine a second set of points mutually agreed drawn randomly from a subset of the field of the rational function we are willing to determine. We will use these random evaluation points to find out if the computed rational function is indeed the desired one with certain probability. Lets call this set R. Now our evaluation point vectors will hold the evaluations of the characteristic polynomials of Q and R.

When a metric is required, the site where the rational function would be determined, say a. receives the evaluation point vector of its counterpart, say b, with $|Q| + |R|$ evaluation points and its cardinality and computes the corresponding divisions. Next, it interpolates a rational function assuming the size of the symmetrical difference is $|Q|$ and tests if this function is in fact the rational function we want with a probability of failure of p. This is done by comparing the divisions that correspond to the points in R against the values of the interpolated function evaluated at those same points. If

the test is successful, then the current interpolated function is reduced and the zeros are obtained. If at least one point fails, then this method is not feasible to determine the differences. Each time the set R of random points is used to evaluate a metric, the sites involved compute another mutually agreed random set. Observe that this can be done asynchronously. Also, since normally R is small, several sets can be maintained simultaneously. Since the interpolation is done once, and $|R|$ is fixed, the computational complexity is $O(|Q|^3)$ and the communication complexity is $|R| + |Q|$. Bearing in mind this complexity we can better estimate the size of Q.

It only remains to show how to compute the size of R from p. Observe that if the values of a key can be mapped to a field of values of size b-bits we can add an additional bit to enlarge the field and we have 2^b more values of size b-bits, where we can take the values of Q and R. Since Q is a fixed set of values, the random values may be chosen from a set of $2^b - |Q|$ values. According to Theorem 4 in [48] and considering the field described above, the probability that two monic rational functions with the sum of their numerator and denominator less than $B = |D_a.R_i| + |D_b.R_i|$ or another upper bound of the number of the symmetrical differences, agree in one randomly selected point although they are different is no more than

$$\overline{p} = (B - 1)/(2^b - |Q|).$$

Observe that since the rational function is constructed from the division of the characteristic polynomial the way it was described, two different rational functions cannot agree in more than the number of points equal to the symmetrical differences of the involved sets minus one. In our case, we took as an upper bound of the symmetrical difference the union of both sets. Observe that if the two rational functions agree at more than this number of points, then they are equivalent. With these ideas in mind, we can determine that the probability that two rational functions are different after agreeing at e consecutive randomly selected evaluation points is no more than $B\overline{p}^e$. Let this probability be p, then the size of R is

$$|R| = \lceil log_{\overline{p}}(p/B) \rceil \geq \lceil log_{\overline{p}}(p/d) \rceil$$

where d is the real number of symmetrical differences.

3.6 Summary

We proposed metrics to discover and quantify referential integrity problems in a distributed database, considering normalized and denormalized schemas. Referential completeness and consistency metrics are defined in a hierarchical fashion at the column, table and database levels. We discussed research issues on distributed query optimization to efficiently compute metrics. Specifically, we studied how to compute our metrics using set reconciliation techniques.

CHAPTER 4

Estimating and Bounding Aggregations

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 [17, 18, 14, 3, 9, 2]. 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 or where exchanging or updating information is frequent, or where table definitions change. In this Chapter we will study the problem of dynamically improve answer sets of aggregation functions in the presence of referential integrity violations.

4.1 Motivating Examples

Our examples throughout this Chapter are based on a chain of stores database with three relations:

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 4.1, where invalid references are highlighted.

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	$\overline{\eta}$	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 4.1: A store database in a relaxed state with invalid foreign keys highlighted.

```
SELECT city.cityId, cityName,
sum(amt)
FROM sales JOIN city ON
     sales.cityId=city.cityId
GROUP BY city.cityId, cityName
UNION
SELECT '-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 4.2: Inconsistent answer set (total is inconsistent).

Example 4.1. The attribute *cityId* in *sales* is a foreign key. The referential integrity constraint, *sales(cityId)* $\rightarrow$ *city(cityId)*, should hold between the two relations. Now observe the query in Figure 4.2. 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 4.3, the answer set is inconsistent in a summarizable sense. That is, using the notation introduced in Section 1.3, 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

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

sum(amt)
316

Figure 4.3: Total sales from valid references (total is inconsistent).

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

as a whole, are inconsistent with respect to the total given by the same query. The answer set represents the original database, but with the tuples with referential integrity errors deleted. Invalid tuples are eliminated from the aggregate function answer set. Now, assuming that there is a high probability that the tuples with invalid foreign keys, that is, invalid values in attribute *cityId*, represent true facts, could we improve the aggregate answer set grouped by *city.cityId*, *cityName*? Can we get an approximate answer set of the true real values?

Example 4.2. Based on the database integrity constraints, the functional dependency: *sales.cityId* $\rightarrow$ *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 4.4. 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 4.1. If we know the functional dependency *sales.cityId* $\rightarrow$ *sales.regionId* holds, could we obtain an improved answer set when grouping by *city.cityId*, *cityName* in the presence of invalid foreign key values? This will be a goal of our approach.

4.2 Aggregations

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

4.2.1 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. As stated in Section 1.3, we will denote as

$R_j[K]$ the set of values in $\pi_K(R_j)$ and may or may not include the null value depending on if η is considered a valid value or not.

The following definition is in the spirit of the definition of partial probability in [47]. 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 4.1. ***Referential partial probability (RPP)*** The RPP is a vector of probabilities that corresponds to a foreign key, say $R_i.K$, where its probabilities are associated to each value of the referenced primary key, say $R_j.K$. Each value corresponds to the probability that an invalid foreign key in $R_i.K$ is actually the associated correct reference in $R_j.K$. Let $k \in R_j[K]$, we say that $R_j.K$ is complete under the RPP if

$$\sum_{k \in R_j[K]} p(k) = 1 \tag{4.1}$$

The idea behind the RPPs is to associate to each primary key value a probability. Each probability corresponds to the probability that the associated value be the correct reference in a tuple with an invalid value in the corresponding foreign key. Notice that for each referenced key, a set of RPPs may be defined, one or more for each foreign key that references it. Users with good database knowledge may assign these probabilities. Depending on the probabilities the user assigns to the valid foreign key values, different RPPs that satisfy completeness (Equation 4.1) can be defined. If the probability values are associated to a discrete probability distribution we can have a uniform or Zipf or geometric or, in general, any probability distribution function RPPs. For example, if all the valid foreign key values were equally probable, a uniform RPP would be defined. Another special case could be a skewed probability distribution function where the user may want to assign probability one to a specific valid reference and zero to all others. Nevertheless, a feasible way to assign these probabilities when computing our proposed aggregate functions is following the intuition that a high probability will correspond to a high frequency in the foreign key and a low probability corresponds to a low frequency.

Definition 4.2. ***Frequency weighted RPP*** Let $k \in R_j[K]$ and $n = |\{ r_i \in R_i \mid r_i[K] \in R_j[K] \}|$, the number of tuples with a valid reference in $r_i[K]$. We define $p(k)$ as

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

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

Example 4.3. Consider the relaxed database of Figure 4.1. The Frequency weighted RPP that corresponds to *sales.cityId* considering the values of the referenced primary key *city.cityId*, $\langle$*LAX,MEX,ROM,MAD,LON* $\rangle$, 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 $\langle$*LAX,MEX,ROM,MAD,LON,* η $\rangle$ the Frequency weighted RPP would be

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

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

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

Definition 4.3. ***Full RPP*** We define $p(k)$ as $p(k) = 1$.

Definition 4.4. ***Restricted RPP*** We define $p(k)$ as $p(k) = 0$.

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

Definition 4.5. ***Referentiality (REF)*** Let $r_i \in R_i$ and $k \in R_j[K]$, we define the referentiality of a foreign key value $r_i[K]$ with respect to k, $REF(r_i[K], k)$, as follows:

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

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 4.1: Extended aggregates according to different RPPs.

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

4.2.2 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. Let $r_i \in R_i$ and $k \in R_j[K]$. Our extendend aggregate functions computed over relaxed databases with referential integrity errors will be defined under a given RPP as follows:

$$x\_count(R_i.PK, r_i[K] = k) = \sum_{r_i \in R_i} REF(r_i[K], k) \tag{4.2}$$

$$x\_count(R_i.A, r_i[K] = k) = \sum_{r_i \in R_i} REF(r_i[K], k) \tag{4.3}$$

$$x\_sum(R_i.A, r_i[K] = k) = \sum_{r_i \in R_i} r_i[A] * REF(r_i[K], k) \tag{4.4}$$

In Equations 4.3 and 4.4, 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 4.1), 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 4.1. Being PK the primary key in R_i, Equation 4.2 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 relation on foreign key-primary key attributes, with potential referential integrity violations.

Example 4.4. Consider the relaxed database of Figure 4.1. Let $s \in sales$. The referentiality of the values in *sales.cityId* that corresponds to each of the valid tuples is shown in Table 4.2.

The referentiality of the same foreign key that corresponds to the invalid tuples using different RPPs is shown in Table 4.3.

Next we show how to compute the different extended aggregates that correspond to the aggregate *sum()* using the corresponding RPP shown in Table 4.3.

Table 4.2: Referentialities of foreign key *sales.cityId* values in valid tuples

$REF(s\,[cityId]\,,k)$	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 4.3: Referentialities of foreign key *sales.cityId* values in invalid tuples with different RPPs

.

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

Frequency weighted referential:

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

$$1 \times 54 + 1 \times 64 + 0 \times 48 + 2/6 \times 33 + 2/6 \times 65 + 0 \times 53 + 0 \times 58 + 0 \times 39 = 150.66$$

Full referential: $f\_sum(sales.amt, s\,[cityId] = LAX) = 216.00$
Restricted referential: $r\_sum(sales.amt, s\,[cityId] = LAX) = 118.00$
Weighted referential: $w\_sum(sales.amt, s\,[cityId] = LAX) = 137.60$
The last expression computed with the uniform RPP.

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

$$\begin{aligned} &x\_max(R_i.A, r_i\,[K] = k) \\ &\qquad = MAX(\{r_i\,[A] * REF(r_i\,[K], k) \mid r_i \in R_i\}) \end{aligned} \tag{4.5}$$

$$\begin{aligned} &x\_min(R_i.A, r_i\,[K] = k) \\ &\qquad = MIN(\{r_i\,[A] * REF(r_i\,[K], k) \mid r_i \in R_i\}) \end{aligned} \tag{4.6}$$

In order to compute the WR and, specifically, the FWR variants, we need to compute the average of the maximum/minimum of the $r_i\,[A]$ values in all R_i of each of the possible ways the valid reference k may be present in the set of invalid tuples. The value that better fits our assumptions as the number of possible ways a given valid foreign key k may be present in the set of invalid tuples is $\binom{n'}{m}$, where $n' = \mid \{\ r_i \in R_i \mid r_i[K] \notin R_j[K]\ \} \mid$ and $m = \lceil p(k) * n' \rceil$, assuming there are invalid tuples. The referentialities of the invalid references are used here to determine the average a correct reference will be present in the invalid tuples. To determine the ways the $r_i[A]$ values of the invalid tuples may be present in this average and obtain the average of the maximum/minimum of $r_i[A]$ of all the R_i instances, we procede as follows. Let $(a_{n'})$ be the sequence of all the $r_i[A]$ values of the tuples in $\{r_i \in R_i \mid r_i[K] \notin R_j[K]\}$ and, in the case of the w_max() functions, the items of $(a_{n'})$ ordered in descending order. In this sequence, the $r_i[A]$ values that meet the condition $r_i[A] < amax$, where $amax = MAX(\{r_i[A] \mid r_i \in R_i \ \wedge\ r_i[K] \in R_j[K]\})$, are then substituted by $amax$. Notice that this sequence may have repeated values.

With these definitions we have

$$w\_max(R_i.A, r_i\,[K] = k) = \begin{cases} MAX(\{r_i[A] \mid r_i \in R_i\}) & if\ n' = 0 \\ \dfrac{\sum_{i=1}^{n'-m+1} a_i * \binom{n'-i}{m-1}}{\binom{n'}{m}} & otherwise \end{cases} \tag{4.7}$$

Intuitively, $(a_{n'})$ has the the maximum values of $r_i[A]$ of all the $\binom{n'}{m}$ instances of R_i where a correct reference may be present in the invalid tuples, ordered in descending order. Thus defined, a_1 will be the maximum of $r_i[A]$ in $\binom{n'-1}{m-1}$ instances, a_2 in $\binom{n'-2}{m-1}$, and so on, up to complete the $\binom{n'}{m}$ instances.

The w_min() functions are defined accordingly, taking sequence $(a_{n'})$ 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 4.5. Consider the relaxed database of Figure 4.1. Let $s \in sales$. To compute $fw\_max(sales.amt, s[cityId] = LAX)$ we procede as follows. The number of invalid tuples is $n' = 2$. The number of tuples that best fit the average of tuples value LAX may be present in these invalid tuples is $\lceil p(k) * n' \rceil = \lceil (2/6) * 2 \rceil = 1$ considering the frequency weighted RPP. The $s[amt]$ values of the invalid tuples are 65 and 33, and the maximum value of $s[amt]$ in the valid tuples is 64. Then $(a_{n'}) = (65, 64)$. According to Equation 4.7 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 1.3, we have the following:

$$x\_count(R_i.PK) = \sum_{k \in R_j[K]} x\_count(R_i.PK, r_i[K] = k) \tag{4.8}$$

$$x\_count(R_i.A) = \sum_{k \in R_j[K]} x\_count(R_i.A, r_i[K] = k) \tag{4.9}$$

$$x\_sum(R_i.A) = \sum_{k \in R_j[K]} x\_sum(R_i.A, r_i[K] = k) \tag{4.10}$$

$$x\_max(R_i.A) = MAX(\{r_i[A] \mid r_i \in R_i\}) \tag{4.11}$$

$$x\_min(R_i.A) = MIN(\{r_i[A] \mid r_i \in R_i\}) \tag{4.12}$$

4.2.3 Function Properties

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

Ascending/Descending, An *ascending* feature as defined in [40] 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 4.1), 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 4.1. *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 4.5, 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 4.2 to 4.5, 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 valid or an invalid reference in $r_i[K]$. As for the WR and FWR variants of the x_max(), Equation 4.7, aggregates, by the definition of the sequence $(a_{n'})$, 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 valid or invalid reference in $r_i[K]$. Deleting a tuple will only possibly decrease the aggregate since, at most, an item of $(a_{n'})$ 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 4.2. *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 the referential integrity errors are repaired (in our context the referential integrity errors are repaired by the substitution of invalid references with correct references without varying the number of tuples) or if there are no referential errors, that is, if the referential integrity constraint holds for all tuples, a *safety* feature holds for the extended aggregations, meaning that the answer sets will not be different compared to the ones from the standard SQL joined aggregations, that is, the SQL grouped attribute aggregations computed over a joined relation on foreign key-primary key attributes. Here, we assume η is invalid.

Proposition 4.3. *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 invalid tuples, $\forall r_i \in R_i, r_i[K] \in R_j[K]$, then, by Definition 4.5 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 relation on foreign key-primary key attributes [36]. □

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 [43]. 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 4.5, and the completeness property of the RPP of these type of aggregates, Equation 4.1.

Proposition 4.4. *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 4.2 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 4.5 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 4.5 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 4.5 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

$$= |\{r_i | r_i[K] \in R_j[K]\}| + |\{r_i | r_i[K] \notin R_j[K]\}| = |R_i| \tag{4.13}$$

□

Summarizable consistency for w_sum($R_i.A$) can be formulated as

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

WR and FWR count(*), count(), sum(), max() and min() total aggregates are invariant wrt referential integrity repairs. As the referential integrity errors are repaired, the total aggregate remains *invariant wrt referential integrity repairs*. That is, the total aggregate remains constant during this type of repair processes. It is easy to see that, for example, for the w_count(*) aggregate, by Equation 4.13 in Proposition 4.4, all the tuples, with or without a valid reference, participate in the total aggregate and since repairing a tuple is equivalent to transfer a tuple from set $\{r_i | r_i[K] \notin R_j[K]\}$ to set $\{r_i | r_i[K] \in R_j[K]\}$, 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 4.1) of the RPP of the WR and FWR aggregates. As for the x_max() and x_min() aggregates, it is easily seen from Equations 4.11 and 4.12, that the total aggregates do not depend on the validity of the foreign key.

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

4.2.4 Assumptions and Probabilistic Interpretation

In order to obtain a valid inference from our extended aggregate functions, it is important that the user bears in mind the following assumptions. Notice we are assuming that $R_j[K]$ is complete. That is, the set of referenced values are all the possible valid values, possibly with the η value, depending on if it is considered valid or not. On the other hand, attribute $R_i.K$ is assumed to have potentially invalid references. Alternative approaches, may consider $R_i.K$ invalid references valid after all, assuming that the error is due to an incomplete set of references in $R_j[K]$.

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

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

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

Following these ideas, we can see that the RR and FR variants of aggregates x_count(*), x_count() and x_sum() are the lower and upper bounds, respectively, of the value the corresponding standard aggregate may take when the referential integrity errors are repaired. The WR and FWR are the expected value, again, of the corresponding standard aggregates and its result depends on which RPP is considered. 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 $\binom{n'}{m}$, (Section 4.2.2), is the total of all the possible ways a valid reference k may be present in the set of invalid tuples, where $n' = |\ \{\ r_i \in R_i \mid r_i[K] \notin R_j[K]\ \}\ |$ and $m = \lceil p(k) * n' \rceil$, assuming there are invalid tuples, then, the meaning of Equation 4.7 in this context is also, as the other WR and FWR aggregates, the expected value of the corresponding standard aggregate.

4.2.5 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 to compute all the plausible answer sets of the aggregate functions, how many are there and does a repair process will eventually give the answer set delivered by the weighted referential aggregations?

Let $e_{R_i.K}$ be the number of tuples in R_i with a referential integrity error in attribute K, that is, $e_{R_i.K} = |\{r_i | r_i \in R_i \wedge r_i[K] \notin R_j[K]\}|$. Also let a *referential integrity repair of attribute* $R_i.K$ be a new instance of R_i, with the same number of tuples, but with the invalid values of attribute K replaced with valid values taken from the set of values of $R_j[K]$. The number of potential referential integrity repairs of attribute $R_i.K$ is $(\mid R_j[K] \mid)^{e_{R_i.K}}$. For our example in Figure 4.1 there are 25 potential referential integrity repairs of attribute *sales.cityId* which is a big number considering there are only 2 referential integrity errors. As for the number of plausible values of a given group once a repair process of a given foreign key have taken place, given the interpretation just discussed we can see that for the aggregate function count() there are $e_{R_i.K} + 1$ or less plausible answers and $2^{(e_{R_i.K})}$ at most for the aggregate function sum(). For our example in Figure 4.1 take the group represented by the value LAX. The plausible answers for the aggregate function count() for the group represented by value LAX are $\{2, 3, 4\}$ since there are 2 invalid references. The aggregation fw_count() gives us 2.6 for value LAX since there are 2 valid tuples with this value, the total number of errors is 2 and, as we saw in Example 4.3, 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 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 4.4 the corresponding value of the fw_sum() for value LAX, *Los Angeles*, is 150.66.

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

```
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 4.5: Query calling fw_sum() and SQL statements evaluating the extended aggregation.

4.3 Extended Aggregations Implementation

In Figure 4.5 we show a query in SQL 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 valid reference in foreign key *cityId*, lines 7 to 26. 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 relation 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*, $freq$;
- finally, the sum() of attribute *sales.amt* for each group, as *sumagg*.

For this implementation we use an alternative way to express fw_sum($R_i.A, r_i[K] = k$). By Equation 4.4 and Definition 4.5 we have:

$$\begin{aligned} &fw\_sum(R_i.A, r_i[K] = k) \\ &\qquad = sum(R_i.A, r_i[K] = k) + sum(R_i.A, r_i[K] \notin R_j[K]) * p(k) \end{aligned}$$

It is easy to see that both fw_count() variants, Equations 4.2 and 4.3, 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

```
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 4.6: SQL implementation of w_sum() with a RPP in table refpp.

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 4.6 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, the corresponding RPP has to satisfy completeness (Equation 4.1). 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 [54] 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 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 reducing the size of the referencing table before joining with the referenced table. In Figure 4.5, to

```
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 4.7: Implementation of early foreign key grouping technique

implement the mentioned technique, the code in lines 8 to 14, may be changed to the one presented in Figure 4.7. In Chapter 5 we present several experiments concerning this optimization.

4.4 Method to Improve FWR Aggregations

We can improve the estimated answer sets of the FWR aggregations if we have another foreign key or another attribute with values of higher quality in the same relation (i.e. an attribute with less referential errors or zero errors). This scenario is possible when two or more databases are integrated and there are relations that share a common primary key. A functional dependency must be defined between the two attributes and the dependency may be in either direction. Although the database may not be in 3NF, remember we are supposing a relaxed database and our goal now is to keep all data, instead of repairing it. Suppose we have in a relaxed database 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 R_{j_a} and *region* for table 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.

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

k_b, of foreign key $R_i.K_b$. That is, a subset of the correct references of $R_i.K_a$, more precisely the following values

$$\{r_i[K_a] \mid r_i \in R_i \wedge r_i[K_a] \in R_{j_a}[K_a] \wedge r_i[K_b] = k_b\}$$

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

Example 4.6. Look at the example of the relaxed database in Figure 4.1. Observe the tuple with value *5* in *sales.storeId*. As discussed before, the frequency weighted RPP may be used to compute how much a value that corresponds to an invalid reference accounts for in the corresponding value of each valid reference. In this case, the value of attribute *sales.amt* that corresponds to the invalid reference η in *sales.cityId* should participate in each valid reference according to the frequency weighted RPP. So far, this is our best estimate. Now, since the user trusts the foreign key *sales.regionId*, the invalid reference mentioned above has a high probability that its 'real' value be a city in the *Americas* region. So the value of attribute *sales.amt 65* should participate only in each valid reference of the *Americas* region.

Let us analyze the case when the user trusts the foreign key $R_i.K_a$. To fix a correct reference value instead of an invalid reference in $R_i.K_b$, we only need to know the functionally dependent value. In both cases we have to consider a relaxed database. That is, we expect referential errors even in the 'trusted' foreign key. Also, the functional dependency constraint may be violated. A feasible approach towards getting better answer sets in the line of the aggregate functions proposed so far is the following.

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

Example 4.7. Again, take for instance the relaxed database in Figure 4.1. Due to the functional dependency *cityId* $\rightarrow$ *regionId*, the valid references of foreign key *regionId* ($\{AM, EU\}$) define the following partition of valid references for foreign key *cityId* in table *sales*:

$$\{\{LAX, MEX\}, \{ROM, MAD\}\}$$

Table 4.4: Cases for foreign key to improve the FWR aggregations - $R_i.K_a \rightarrow R_i.K_b$, $r_i \in R_i$

	$\mathbf{R_i.K_a}$	$\mathbf{R_i.K_b}$	$\mathbf{R_i.A}$ values or tuples grouped by
1	$k_a \in \{r_i[K_a] \mid r_i[K_a] \in R_{j_a}[K_a]\}$ k_a is valid	$k_b \in \{r_i[K_b] \mid r_i[K_b] \in R_{j_b}[K_b]\}$ k_b is valid	k_a
2	$k_a \in \{r_i[K_a] \mid r_i[K_a] \notin R_{j_a}[K_a] \wedge r_i[K_b] = k_b\}$ $\exists k \in \{r_i[K_a] \mid r_i[K_a] \in R_{j_a}[K_a] \wedge r_i[K_b] = k_b\}$ k_a is invalid $\exists$ valid ref. in subset by k_b	$k_b \in \{r_i[K_b] \mid r_i[K_b] \in R_{j_b}[K_b]\}$ k_b is valid.	$k \in \{r_i[K_a] \mid r_i[K_a] \in R_{j_a}[K_a] \wedge r_i[K_b] = k_b\}$
3	$k_a \in \{r_i[K_a] \mid r_i[K_a] \notin R_{j_a}[K_a] \wedge r_i[K_b] = k_b\}$ $\nexists k \in \{r_i[K_a] \mid r_i[K_a] \in R_{j_a}[K_a] \wedge r_i[K_b] = k_b\}$ k_a is invalid $\nexists$ valid ref. in subset by k_b	$k_b \in \{r_i[K_b] \mid r_i[K_b] \in R_{j_b}[K_b]\}$ k_b is valid.	$k \in \{r_i[K_a] \mid r_i[K_a] \in R_{j_a}[K_a]\}$
4	$k_a \in \{r_i[K_a] \mid r_i[K_a] \in R_{j_a}[K_a]\}$ k_a is valid	$k_b \in \{r_i[K_b] \mid r_i[K_b] \notin R_{j_b}[K_b]\}$ k_b is invalid	k_a
5	$k_a \in \{r_i[K_a] \mid r_i[K_a] \notin R_{j_a}[K_a] \wedge r_i[K_b] = k_b\}$ k_a is invalid	$k_b \in \{r_i[K_b] \mid r_i[K_b] \notin R_{j_b}[K_b]\}$ k_b is invalid	$k \in \{r_i[K_a] \mid r_i[K_a] \in R_{j_a}[K_a]\}$

We can define two *frequency weighted RPPs*, considering the two relations, each one of them with the tuples that have the values of each of the above subsets in foreign key *cityId*. Taking relation *sales* in Figure 4.1 the corresponding vectors are:

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

Contrast these two vectors with the vector shown in Example 4.3.

Table 4.4 shows the cases that should be considered depending on the values of $R_i.K_a$ and $R_i.K_b$ in each tuple and how should a $R_i.A$ value or tuple participate in an aggregate function in order to improve the estimated aggregate answer sets preserving the *summarizable consistency*, *ascending/descending* and *safety* properties in the aggregate functions where these properties apply.

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

Table 4.5: Referential aggregate sum(), FWR and FWR improved with trusted foreign key *regionId*. Both frequency weighted RPP are shown.

cityId	cityName	sum(amt)	fw_sum(amt)	fw_sum(amt) improved	Freq. w. RPP	Freq. w. RPP improved
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

can reconstruct the functional dependency by choosing for each correct reference k_a in $R_i.K_a$ the correct reference k_b in $R_i.K_b$ to which k_a is associated the most. Ties are solved simply choosing one value. According to Table 4.4, if there is not a correct reference k_b in $R_i.K_b$ for k_a, then the tuples with a k_a reference in $R_i.K_a$ belong to case 4. If there are tuples with a k_a in $R_i.K_a$ associated to a correct reference k_b in $R_i.K_b$, but this pair of values was not the maximum pair of values for value k_a then these tuples will be treated as belonging to cases 2 or 3 since the user trusts foreign key $R_i.K_b$ so we assume k_a is an error. Take the database in Figure 4.1. We show in Table 4.5 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 correct reference of foreign key K_a determines only one value of K_b. If a pair of correct values k_a, k_b have not the maximum frequency, reference in attribute K_b will be considered an invalid value.

4.5 Summary

We improved SQL aggregations to return enhanced answers sets in the presence of referential integrity errors. Referential integrity errors are treated as imprecise values that stand for precise values, determined by a foreign key. We proposed two families of extended aggregate functions: weighted referential (WR) aggregations and full referential (FR) aggregations. The definition of these extended aggregate functions is based on a new concept named referentiality. Intuitively, referentiality is the degree to which a foreign key value in a tuple that belongs to the referencing table, refers to a correct reference in the referenced table. Extended aggregations represent a complement to standard SQL aggregations and they are studied under a common probabilistic framework. WR aggregations are based on referential partial probability vectors (RPPs) associated with the foreign key. A particular family of the 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. Full referential aggregations present an extreme repair scenario where each aggregated group receives all the values corresponding to existing referential integrity errors. Full referential aggregations are helpful when the user needs to include for each group all tuples with invalid references. Our extended aggregations exhibit important properties, which are essential to consider them as correct extensions of standard SQL aggregations. A WR aggregation for row counts is summarizable consistent, ascending and safe. 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 of the mentioned aggregates, together with WR and FR max and min aggregates, their total aggregate share the invariant with respect to referential integrity repairs property. The max extended aggregates are ascending and min aggregates, descending. Both fulfill the safe property. On the other hand, FR aggregations are safe and plausible. The latter property means the answer set represents a potential repair for each group, that consists in assigning to all invalid references, the reference that represents each group with a valid key. Equivalent SQL expressions are given to compute our extended aggregates. Our proposal also includes a method to improve answer sets taking advantage of other related attributes via a functional dependency. Specifically, we analyze the case when such attributes are also foreign keys.

CHAPTER 5

Experimental Evaluation

We present an extensive experimental evaluation of our referential integrity local QMs and of our extended aggregates with real and synthetic databases. We used five real databases and one synthetic database, generated with the TPC-H DBGEN program. All times are reported in seconds, unless stated otherwise.

We used multiple relational DBMSs, from different software companies. To avoid discussion on each DBMS specific features, performance and SQL compliance, we omit their real name. This was also a request from end users to protect their privacy, while allowing us to report our findings. We used standard SQL (ANSI) in our QMs implementation, making it portable in all relational DBMSs.

5.1 Referential Integrity QMs in Real Databases

Real databases were used to assess the usefulness of our approach and the relative importance of QMs. In the experiments presented below we asked several information technology organizations to give us permission to discover referential integrity problems in their databases. The real databases came from a university, a government organization and a retail company. In our discussion we change the actual table and column names to protect privacy. Organizations did not agree to reveal attribute value QMs since they revealed specific information about their operation. In general, query running times are given in seconds.

Educational Institution

We now present interesting results on a database from a public higher educational institution. Since this was a production environment, other database processes were

Table 5.1: Educational institution: attribute level QMs.

R_i	K	*rcom*	time
student	studentId	19.0%	707

Table 5.2: Educational institution: attribute QM statistics.

R_i	K	min	μ	max	σ
student	studentId	1	3	9	1.52

running concurrently with our program, but the workload was similar in all cases; we report the average running time. Due to security restrictions we could only explore one important historic table containing important student information, including student name, year, semester, course names, course grades, credits and grade point average. We were not allowed to explore value level QMs, showing specific referential errors for specific students. The experiment goal was to validate that student identifications were actually valid, according to a reference table containing biographic and demographic information for all students ever registered at the institution. Results were discussed with the Information Technology manager, two database developers and a system administrator. Results are presented in Table 5.1. The IT manager was aware there existed some referential integrity issues. Database developers were surprised there was such a high fraction of missing foreign keys for student identification numbers. The system administrator stated the historic table was constantly updated with new semester course grades, but he said this table had a few changes in its definition in the past. Table 5.2 gives statistical information about the probabilistic distribution of invalid references. It is noteworthy that there exists a value that contributes significantly to $acom()$ with 9 invalid references. The coefficient of variation σ/μ states there is not much variation around μ: most values contribute equally to $acom()$.

Government Organization

Applying our approach on a different database, we now present QMs on a driver's license database from a state in Mexico (state name omitted due to privacy restrictions). This database contained driver's license information and vehicle registration information for people living in certain counties. The database had historic tables containing historic personal information, driving records, traffic fines and payments. There were additional reference tables containing personal identifiers issued by government, similar to the US Social Security Number. These results were obtained during an early stage of our project, where we did not compute all QMs. Results were discussed with

Table 5.3: Government database; database level QMs.

$\mathcal{R}$	N	$rcom$	$rcon$
GovtDB	23	0.43%	0.01%

Table 5.4: Government database; relation level QMs.

R_i	k_i	n_i	$acom$	$rcom$
docum	3	4779940	2081465	14.52%
genId	3	2493855	15198	0.14%

the IT manager, a database applications developer and a database administrator. The database level QMs are shown on Table 5.3, where we can see there are minor referential integrity problems. Incompleteness of foreign keys is clearly more important than consistency, even though the database is denormalized. This was good news for the IT manager, who did not anticipate having any important referential issues. Going down one level, Table 5.4 shows a couple of relation level QMs. Users became aware referential problems were prevalent specifically in relation *docum* which was the most important relation being continuously updated. To a lesser extent, there were both completeness and consistency problems in relation *genId*. Then going to a more granular storage level, attribute level metrics are shown in Table 5.5. QMs $a()$ and $r()$ are significantly high for the *county* attribute. On the other hand, our prototype uncovered inconsistency problems in attributes *brName* and *model*. Before computing our QMs, users had told us there could be referential violations. Nevertheless, they had not imagined there were serious problems in many cases.

Table 5.5: Government database; attribute level QMs.

R_i	A_{ij}	*com or con*	$a()$	$r()$
docum	county	*com*	2076530	43.443%
docum	citizen	*com*	4935	0.103%
docum	paymts	*com*	0	0.000%
genId	brName	*con*	5628	0.226%
genId	model	*con*	4562	0.183%

Table 5.6: Retail database; attribute-level QMs.

Attribute	*com or con*	$a()$	$r()$
storeId	*com*	9468	7.56%
format	*con*	9672	7.72%
region	*con*	15036	12.01%

Table 5.7: Retail database; $a()$ error correlation.

ρ	$acom(storeId)$	$acon(format)$	$acon(region)$
$acom(storeId)$	1.000		
$acon(format)$	0.988	1.000	
$acon(region)$	0.774	0.764	1.000

Retail Company

We present error correlation results with a database from a retail company. We focused on a summary fact table, having $n_i = 125,208$ rows, used to perform store-level data mining analysis and to build monthly and annual OLAP reports. This fact table had been built from a database containing 6 billion transactions. From an analytical perspective this was one of the most important tables to perform OLAP and data mining analysis for this company. We ran our QMs and they took just 10 seconds; the correlation matrix was derived in 2 seconds. Table 5.6 shows attribute-level QMs only for those attributes with non-zero error. Clearly, completeness is a major issue since more than 7% of rows have an invalid FK for *storeId*. Looking at foreign attributes we can see $acon(format)$ is close to $acom(storeId)$ which indicates we cannot tell if $format$ is consistent or not with the value in the referenced relation; given our conservative definition we assume it is incorrect. On the other hand, $acon(region)$ is far from $acom(storeId)$, which reveals a serious consistency problem. In fact, 15036-9468=5568 values are inconsistent despite the fact that the FK *storeId* exists. Table 5.7 shows a high correlation between *storeId* and *format*, which is a consequence of the functional dependency. However, *region* shows a lower correlation to either attribute, which tells us that this attribute shows inconsistency in an independent manner. In practical terms, this means repairing referential errors in *storeId* will take care of *format*, but *region* will require a separate (independent) repair action.

Users Feedback Summary

In general, users expected their databases to be clean and they asked us to keep their specific organization names and specific record information confidential. In some cases, they also requested to keep the DBMS brand confidential as well. We requested getting access to critical databases that were updated and refreshed continuously. It did not matter if such databases were denormalized since our program was prepared to handle them. In particular, we focused on analyzing large historic tables whenever possible since those tables are used in a wide variety of queries and reports. Under the IT manager supervision we were allowed to compute our QMs with automatically generated SQL queries. When the referential integrity prototype had generated results, these results were discussed with the IT managers, database developers and database administrators. We presented results hierarchically going from the database level QMs down to attribute level QMs. Based on findings, users requested to browse a sample of records with referential errors. Some users were intrigued by results and asked us to explain how our QMs were computed with SQL queries.

We asked the IT managers the following questions. Are you surprised QMs indeed uncovered some referential problems?, what level of granularity of QMs would you compute on a frequent basis?, which is a more critical dimension in your database: completeness or consistency?, for which tables is it critical to maintain referential integrity? We asked database developers and database administrators the following questions. Why do you think table X has referential errors?, when table X is denormalized, how is it computed?, how frequently is database X updated?, is table X updated in batch or continuously, inserting one record at a time?, is there an explanation for attribute Y to have high levels of referential errors, compared to other attributes?

In general, users feedback about our tool was positive. Users stated that QMs helped them to discover unexpected referential integrity problems and to ensure data quality policies and procedures were working properly. Since most tables were normalized, QMs on foreign keys were more interesting. Database level QMs were not particularly useful on the three real databases because relative error was high. Attribute level QMs on foreign keys (completeness) were particularly useful in OLTP systems or isolated databases in which referential integrity was routinely enforced. That is, completeness was more important for databases where records were inserted by transactions. Attribute level QMs on foreign attributes (consistency) were valuable to identify stale records in large fact tables and to detect inconsistent attributes in denormalized tables used for data mining purposes. That is, consistency was more important for a data warehouse where there exists a large fact table with historic information and where there are denormalized tables computing aggregations from the fact table. Database and relation level QMs helped detecting unforeseen referential integrity issues. Attribute and value level QMs helped diagnosing (explaining) referential errors and preparing a repair plan. Users feedback is summarized in Table 5.8.

We now present a more detailed discussion of users comments. A system manager

Table 5.8: Users feedback.

QM level	usefulness	application	user
database	low	detection	IT manager
relation	medium	detection	IT manager/DBA
attribute	high	diagnosis	DBA/developer
value	high	diagnosis/repair	DBA/developer

stated that QMs revealed problems he was not aware of, and another IT manager stated that QMs helped him obtain a clear idea about data quality. A DBA for a data warehouse stated that QMs could enrich metadata to test data loading scripts in order to detect problems while tables are being refreshed. An IT manager said QMs could justify a plan to improve data quality when integrating databases into a central data warehouse. QMs provided varied usefulness depending on each type of user. QMs at the relation and database levels were more useful for IT managers since they give a high level referential integrity quality assessment. QMs at the attribute and value level were more useful for database application developers, who suggested integrating QMs with testing database application code. FK value level QMs (invalid FK frequencies) were interesting to all users (e.g. a FK appearing in many relations) because they provided evidence about problems, but users stated they preferred to run the prototype on their own. An IT manager and a DBA stated that QMs should be collected over a long period of time (e.g. every week), especially for large historic tables containing transaction data, to track data quality and prevent future data quality problems.

5.2 Extended Aggregations in Real Database

We present the use of our extended aggregations in two real databases.

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 loosing 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, refere to Section 4.4, 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. 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.

Retail Company

An important retail company in Mexico listing at the stock market since more than 25 years ago, tried our extended aggregates in one of its applications to assess the usefulness of our approach. Our aggregates were used in a reward program applied to agents. The agents are advisers working in specialized departments that participate in sales and they earn commissions based on sales depending on the number of sales and the total amount sold of their corresponding products. In every point of sale, a seller

Table 5.9: Transaction detail in a given point of sale (p. of s.).

date	store	p. of s.	transId	clientId	agentId	productId	amt	sellerId
02/23/07	2..	5	1276	44...547	14545779	147409814	$22,347.83	73...03
		5	1277	44...547		147409821	$16,086.96	73...03
		5	1278	15...577	12017425	145215677	$1,477.39	73...03
		5	1279	21...430	12017378	145215684	$2,826.09	73...03
		5	1280	21...430		145577225	$773.91	73...03
		5	1281	43...940		145577218	$513.04	73...03
		5	1282	23...948		145573784	$4,513.04	73...03

Table 5.10: Total commission per agent and bonus computed using fw_sum()

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

date	agentId	amt.	comm.	sales	bonus
02/23/07	14545779	$282,231	$28,223	12	$1,760
02/23/07	12017425	$438,111	$43,811	13	$1,906
02/23/07	12017378	$138,168	$138,16	9	$1,320
03/14/07	15084536	$333,949	$33,394	17	$2,493
03/14/07	15213754	$171,440	$17,144	14	$2,053
03/14/07	12017033	$180,504	$18,050	10	$1,466

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 5.9 shows several registers of how the information is received, some of them with no information in the agent's code field, *agentId*.

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. Table 5.10 shows the amount of sales per agent, the commission earned and the bonus computed using fw_sum().

Table 5.11: 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	$(1-p)^{n-1}p$	$p = 1/2$
Normal	$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	$\mu = 3000$ $\sigma^2 = 1000$

5.3 Synthetic Databases

We conducted our experiments on a database server with one Intel Xeon CPU at 1 GHz with 256 MB of main memory and 108 GB on disk. The relational DBMS we used was Postgres Version 8.0.1.

Our synthetic databases were generated by the TPC-H DBGEN program, [64], with scaling factors 1 and 2. We did not define any referential integrity constraint to allow referential errors. We inserted referential integrity errors in the referencing fact table (*lineitem*) with different relative errors (0.1%, 0.2%,..., 1%, 2%,..., 10%). The invalid values were inserted following several different probability distribution functions (pdfs) including geometric, uniform, zipf and normal, and in three foreign keys (*l_orderkey*, *l_partkey* and *l_suppkey*).

The results we present in this section, unless stated otherwise, use a default scale factor 1. The referencing table, *lineitem* and the referenced tables, *orders*, *part* and *supplier* have the following sizes: 6M, 1.5M, 200k and 10k tuples, respectively. Invalid FK values were randomly inserted according to four different pdfs, as shown in Table 5.11. Elapsed times are indicated in seconds.

Query Optimizations to Evaluate QMs

First, we evaluated the left outer join optimization on foreign keys, summarized in Table 5.12. Performance degrades significantly for the set containment query, as the cardinality of the referencing relation increases. On the other hand, it is noteworthy time grows more slowly for the left outer join query. To explain why set containment queries are slower than left outer join queries, we obtained the query plans for both types of queries on two DBMSs. We found that the query optimizer in DBMS X first produced a sequential scan for the nested subquery and then a nested loop join to determine the negated set containment. The optimizer from DBMS Y produced a nested loop join for the same query, which was more efficient. On the other hand, the

Table 5.12: Query optimization: left outer join and set containment

n_i	Left outer join	Set containment
10,000	25	60
20,000	25	240
30,000	29	600
50,000	35	1560
100,000	37	6120

Table 5.13: Query optimization: early vs. late foreign key grouping.

n_i	Late	Early FK grouping	
	FK group	Size 4	Size 5
1'200,000	54	67	56
2'500,000	91	136	85
5'000,000	165	172	134

left outer join was generally computed with a merge sort or hash join. These results supported using a left outer join by default.

We compared the performance between the late and the early foreign key grouping optimization for small groups of invalid foreign key values. If the number of distinct values in the foreign key was similar to the relation cardinality (i.e. large) applying early foreign key grouping was counterproductive. But if the number of distinct values on foreign keys was smaller, then this optimization produced a significant speedup. In Table 5.13 we present performance for different cardinalities, considering foreign key groups of size 4 and 5 (meaning each invalid value appeared in 4 or 5 tuples on average). Times become better for group size 5. Small referenced relations or large referencing relations having few distinct FK values make early foreign key grouping an essential optimization, since it significantly reduces the size of the relation to be joined with all its referenced relations.

Figure 5.1 analyzes the impact of referential errors on time performance with two representative pdfs. We can see $a()$ has a marginal impact on performance as it increases. The impact is more important for the zipf distribution. Summarizing, the size of a relation is far more important than the number of invalid values.

Statistics on QMs

Table 5.14 shows a summary of statistics of QMs with a fixed $r() = 1\%$ for the *lineItem* table and the *partId* column. Observe that for the geometric pdf one invalid value pro-

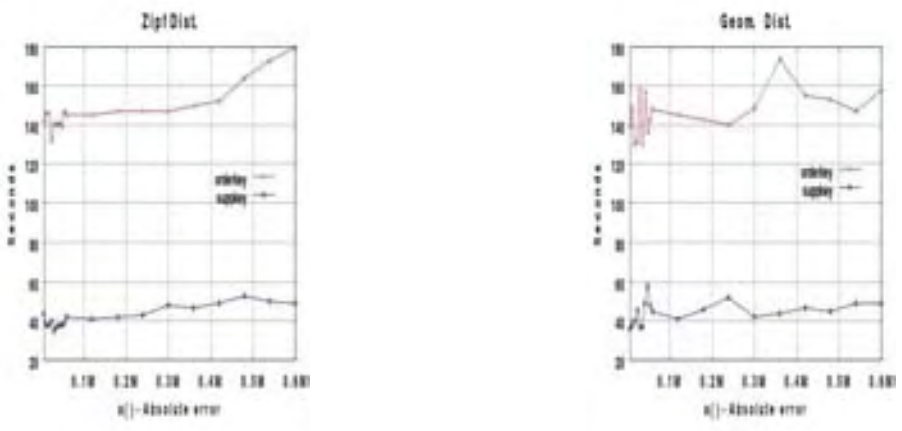

Figure 5.1: Early FK grouping variant with two pdfs.

Table 5.14: Univariate statistics with different pdfs.

pdf	min	μ	max	σ
Uniform	1	10	22	3
Zipf	1	7	6002	82
Geometric	1	3505	29873	7825
Normal	1	120	372	121

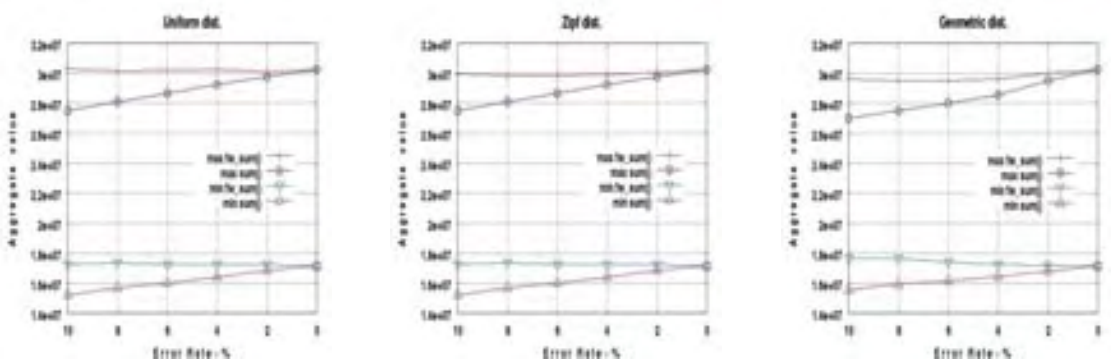

Figure 5.2: Accuracy of the fw_sum() aggregate function.

duces many referential errors. On the other hand, for the uniform pdf all invalid values contribute evenly to referential errors. The geometric pdf has the highest standard deviation, meaning there are values far from the mean, that significantly contribute to error.

Approximation Accuracy of Extended Aggregations

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 uniform, Zipf and Geometric 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 when needed. We simulated a process of gradually repairing the database and within this process we also computed our proposed aggregate functions. Remember that in our framework, we are not interested in how repairs are done, but in getting an approximation of a complete answer set. We then evaluated the 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 5.2 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

Table 5.15: 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())

(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 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 5.15), 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 5.15, 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.

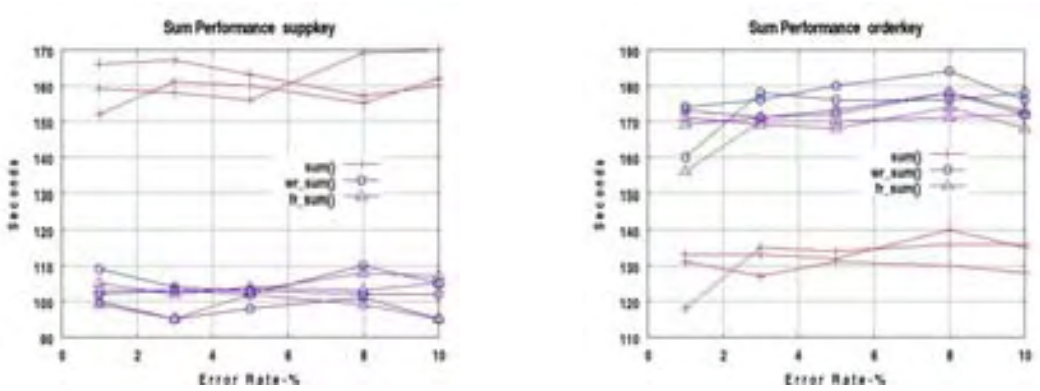

Figure 5.3: Comparing time performance of aggregations.

Time Performance of Extended Aggregations

The queries used to compute the WR, FWR and FR aggregations first compute an auxiliary table, fw_temp in Figures 4.5 and 4.6. 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 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 5.3 for the aggregate functions fw_sum() and fr_sum(). Our experimental results evaluate performance of extended aggregation against standard SQL joined aggregations with foreign keys $l\_orderkey$ and $l\_suppkey$ of relation $lineitem$, 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 5.3. 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 5.16 we can see the time it took to compute

Table 5.16: Time performance computing auxiliary table with several referenced tables of different sizes and using *lineitem* 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

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 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.

CHAPTER 6

Related Work

There is extensive work on maintaining referential integrity. For instance, [46, 51] identify conditions to avoid referential problems during data manipulation. Reference [35] presents a model for referential integrity maintenance during run-time execution. There is a proposal to check deferred referential integrity using a metadata network [15], where inclusion dependencies are considered and each foreign key value is verified to have a corresponding matching key value. No analysis is done on measuring inconsistency, nor on algorithm performance. Implementation of update and delete referential actions, full and partial types, reference types, are features that have not been fully completed in commercial DBMSs [65].

The SQL language has supported referential integrity by defining foreign keys and referential actions. A survey on SQL query optimization is given in [16], explaining when to perform a "group-by" operation before a join operation (early or eager), instead of joining tables first and then performing the "group-by" operation (late or lazy). This optimization is applicable when the number of result groups is small and when a different evaluation order does not change result correctness. Our early foreign key grouping is a particular case of this optimization, but we also consider null key values and we generalize it to build a cube of foreign keys. SQL aggregations are extended to return approximately consistent answer sets when there exist invalid FKs [53]; this approach dynamically detects referential errors and improves answer sets in two ways: (1) by distributing aggregated values in a measure attribute from invalid FK values among valid FK values; (2) by exploiting valid FK values in another attribute to make distribution more accurate.

Referential Integrity QMs

To our knowledge, data quality metrics have not been proposed with respect to referential integrity. A proposal of simple metrics for data quality in relational databases is given in [50], where completeness and soundness metrics are introduced. These metrics compare the state of the database to the data from the real world such database is supposed to represent. Our proposed QMs measure the completeness of references among relations and to some extent QMs on foreign attributes measure soundness. We do not deal with the problem of measuring if a valid reference is indeed valid. Concerning this aspect, [66] introduces an attribute-based model that incorporates data quality indicators and focuses on two dimensions: interpretability and believability. In [62] the authors introduce an algebra to measure completeness in several data quality dimensions, considering basic relational set operations, null values and the open/closed world assumptions, but ignoring referential integrity.

A model linking data quality assessment to user requirements is proposed in [27]. In [59] a classification of objective quality measures is presented. In particular, their simple ratio function measures the ratio of desired outcomes to total outcomes. The authors suggest that consistency could be measured by these means. This is related to our referential integrity error definitions. In [45] the authors investigate the correct integration of relationship instances integrated from different source databases, focusing in detecting semantic conflicts and reconciling them. This proposal can be used as pre-processing to get QMs, since semantic conflicts must be solved before quantifying referential integrity errors. Referential integrity metrics have been studied considering the model design point of view. Reference [12] introduces two metrics to aid model designers make better decisions by analyzing referential paths and the number of foreign keys in a schema. The authors do not consider invalid keys or denormalized relations. In contrast, our approach is applicable after the database logical data model has evolved from its original design or for database integration. Authors in [63] introduce metrics to evaluate the effectiveness of conceptual models for a data warehouse, which complement logical and physical design tools.

We now explain our approach from a broader perspective. Data quality problems related to referential integrity are described in [37]. The authors distinguish between operational and diagnostic data quality metrics. In this work referential violations between two tables are called poor join paths. Our metrics at the database and relation levels are diagnostic and frequencies of invalid foreign key values are operational since they provide insight into how to fix referential errors.

Healthcare data warehouses represent a prominent example, where data quality (missing information, inconsistency), data integration from diverse sources (structured and semistructured) and privacy (confidentiality of patient records) make database management difficult [6]; several of such issues are related to referential integrity maintenance among tables coming from different sources. On a closely related topic, [5, 26] study data quality issues in data warehouses considering the time dimension for aggre-

gate queries; in our work we simply detect referential integrity errors on the current state of the database, but we do not track when referential errors were introduced. Therefore, discovering and explaining referential errors back in time is an interesting issue for future work.

Discovering database relationships and repairing inconsistent databases are important related problems. In [24] the authors propose techniques to automatically identify PK/FK relationships between tables coming from different databases; in our case we assume such relationships are manually specified by the user or come from a logical data model. Therefore, this approach can be applied as a pre-processing phase before computing referential integrity QMs. In [7] the authors introduce a cost framework to aid in the restoration of a database with integrity constraints (user-defined rather than referential) violations via value modifications. Constraints in an inconsistent database can be satisfied by incorporating constraint checks in equivalent rewritten queries to given queries [29]. Both approaches apply heuristics to repair databases, either statically (by updating tables) or dynamically (by rewriting queries). In contrast, our work diagnoses problems and gives detailed information to the user in order to explain and repair referential errors. Therefore, our proposal can serve as a pre-processing step before applying any heuristic to repair invalid values.

Our distributed QMs extend quality metrics defined on a single database [54]. We generalized a local database state and studied the problem from a distributed perspective. Also, our distributed QMs identified new issues related to distributed query optimization. Measuring replica consistency in distributed databases is an important related problem. The replication has been proposed in several ways such as horizontal or vertical fact replication, complete or partial dimension table replication [22]. It is common that all these techniques turn the systems complex and difficult to manage [1]. Replica consistency has received much attention in recent years. In [58] the authors propose two update propagation strategies that improve freshness, a concept that supposes that replica consistency in a distributed database can be relaxed. These strategies are based on immediate propagation, without waiting for the commitment of the update transaction in Master-Slave configurations. In [57] the authors propose a refreshment algorithm to maintain replica consistency in a lazy master replicated database based on specific properties of the topology of replica distribution across nodes. Both works propose strategies towards maintaining replica consistency in a database in a Master-Slave configuration. Our work supposes an a posteriori scenario where inconsistency is probably present and with our *gcur* metric the user wants to measure it. Our work is oriented towards highlighting the benefits to use our methods in a distributed database, in a Master-Master configuration. In [8] the authors propose two lazy update protocols that can be used in a distributed data warehouse, that guarantee serializability but require that the copy graph be a directed acyclic graph. The authors propose a solution to prevent the lazy replication inconsistency problems in a particular distributed configuration.

In [30] the authors introduced a coherency index to measure replica consistency (co-

herency). They examine the trade off between consistency and performance, and show that in many situations a slight relaxation of coherency can increase performance. In our work, we focus on measuring the quality of the database with respect to replica and referential integrity consistency and evaluate how to obtain a fast diagnosis considering different replica scenarios.

In [69] and [70] the authors proposed several metrics to measure the quality of replicated services where the access to a replicated database is included. They show a middleware layer that enforces consistency bounds among replicas allowing applications to dynamically trade consistency for performance based on the current service, network, and request characteristics. They measure availability while varying the consistency level, the protocol used to enforce consistency, and the failure characteristics of the underlying network. However they measure the quality of the service (access) and not the quality of, as in our case, the replicated data.

In [42] the authors propose an approach to repair a crashed site in a distributed data warehouse that uses data replication to tolerate machine failures. Their approach uses timestamps to determine which tuples need to be copied or updated. Our repair strategy is an on demand technique, that also queries sites but does not require timestamps. It is based in the efficient computation of the symmetrical differences among replicas.

Extended Aggregations in Databases with Referential Integrity Errors

Research on managing and querying incomplete data has received significant attention. In [17] 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 [21], 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 extended aggregates, 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 [14]; 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 [13] the authors identify two complementary frameworks to define views over integrated databases and they propose techniques to answer SPJ queries where there are missing foreign key values; the authors prove the problem of getting consistent answer sets is significantly difficult (non-polynomial time). In [3] 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 [29] based on query rewriting, the authors proposed a system named ConQuer that retrieves data that is consistent wrt key constraints given by the user together with their queries. A similar strategy is used in [33], but for consistent answering of conjunctive queries under key and exclusion dependencies. This is done by rewriting the query in Datalog with negation. In [19] 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 which 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 [47], 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 [47], but uncertain value should be stored. Also, associated to the values of the referenced primary key with respect to a given foreign key, there is a vector that holds for each value of the primary key, the probability that this value appears in a given tuple in the foreign key of the referencing relation. Users can assign these probabilities, but we give a feasible and automated method, exploiting the frequency weighted RPP, to get such probabilities. Reference [9] 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 [10] 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 [11] 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.

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 [39]. 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.

Concerning the properties of the aggregate functions, in [40] 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 [43], 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 [28] 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 [71] the author uses logic theories based on a probabilistic first order language to formalize probabilistic databases. In [41] 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 [23] 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 [61] 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* [67], a DBMS for uncertain and probabilistic data, in [52] 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.

CHAPTER 7

Conclusions and Future Work

We proposed a comprehensive set of centralized and distributed quality metrics (QMs) for referential integrity, which can be applied in data warehousing, database integration and data quality assurance. Our QMs measure completeness in the case of foreign keys and consistency in the case of foreign attributes in denormalized databases. QMs are hierarchically defined at four granularities: database, relation, attribute and attribute value. Quality metrics are of two basic types: absolute and relative error. Absolute error is useful at fine granularity levels or when there are few referential violations. Relative error is adequate at coarser granularity levels or when the number of referential violations is relatively large. We introduced univariate and bivariate statistics on attribute level QMs to further understand the probability distribution of invalid foreign key values.

We improved aggregations to return enhanced answers sets in the presence of referential integrity errors. Referential integrity errors are treated as imprecise values that stand for precise values, determined by a foreign key. We proposed two families of extended aggregate functions: weighted referential (WR) aggregations and full referential (FR) aggregations. The definition of these extended aggregate functions is based on a new concept named referentiality. Intuitively, referentiality is the degree to which a foreign key value in a tuple that belongs to the referencing table, refers to a correct reference in the referenced table. Extended aggregations represent a complement to standard aggregations and they are studied under a common probabilistic framework. WR aggregations are based on referential partial probability vectors (RPPs) associated with the foreign key. A particular family of the 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. Full referential aggregations present an extreme repair scenario where each aggregated

group receives all the values corresponding to existing referential integrity errors. Full referential aggregations are helpful when the user needs to include for each group all tuples with invalid references. Our extended aggregations exhibit important properties, which are essential to consider them as correct extensions of standard aggregations. A WR aggregation for row counts is summarizable consistent, ascending and safe. 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. With these properties, we want to assure that the user receives consistent answer sets. All of the mentioned aggregates, together with WR and FR max and min aggregates, their total aggregate share the invariant with respect to referential integrity repairs property. The max extended aggregates are ascending and min aggregates, descending. Both fulfill the safe property. On the other hand, FR aggregations are safe and plausible. The latter property means the answer set represents a potential repair for each group, that consists in assigning to all invalid references, the reference that represents each group with a valid key.

We explained how to efficiently calculate QMs and extended aggregates with SQL queries. Specifically, we presented two query optimizations. The first optimization favors a left outer join over a set containment to use a hash or merge-sort join algorithm instead of a nested loop algorithm. The second optimization performs a group-by operation on foreign keys before a join (pushing aggregation, early group-by) to reduce the size of the referencing relation. This optimization is effective for large relations with many foreign keys, where the number of distinct values per foreign key is small. Experiments evaluate referential integrity QMs and extended aggregates with real and synthetic databases on different DBMSs. We got interesting results on real databases and end-users opinion was positive. QMs at the database and relation level were more useful for managers, whereas value and attribute level QMs were more interesting to DBAs and application programmers. We studied quality metrics for attributes following four different probability distributions. Attribute values with a high relative error in skewed distributions can be used to fix critical referential problems. Univariate statistics and correlations can help understand the probability distribution and co-occurrence of referential errors. On the time performance side, a left outer join evaluation was generally more efficient than a set containment due to fast join evaluation algorithms in different DBMSs. On the other hand, early foreign key grouping was always more efficient than late foreign key grouping. Our experiments with our extended aggregates show the answer sets returned are consistent approximations and they also show the overhead due to additional computations is reasonable.

Our work can be extended to repair a database considering the frequency of matching values for foreign keys and foreign attributes. Our statistics on attribute level metrics can be extended to apply multidimensional data mining techniques, such as clustering and factor analysis. We want to study how to efficiently repair a denormalized database to leave it in a strict state, based on a plan derived from quality metrics. Incremental computation is needed to keep quality metrics up to date in ever-growing

data warehouses. Finally, we believe our ideas may be applicable in semistructured data, such as text or XML.

Some of our ideas related to our extended aggregates can be extended to more general SPJ queries, especially involving multiway joins. We want to improve the definition of WR aggregations to consider correlation among attributes. We would like to study the alternative scenario where the referenced relation 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.

Bibliography

[1] J. Albrecht and W. Lehner. On-line analytical processing in distributed data warehouses. In *IDEAS '98: Proceedings of the 1998 International Symposium on Database Engineering & Applications*, page 78. IEEE Computer Society, 1998.

[2] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 68–79. ACM, 1999.

[3] 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.

[4] 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.

[5] D.J. Berndt and J.W. Fisher. Understanding dimension volatility in data warehouses. In *INFORMS CIST Conference*, 2001.

[6] D.J. Berndt, J.W. Fisher, and J. Studnicki A.R. Hevner. Healthcare data warehousing and quality assurance. *IEEE Computer*, 34(12):56–65, 2001.

[7] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *ACM SIGMOD Conference*, pages 143–154, 2005.

[8] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, and A. Silberschatz. Update propagation protocols for replicated databates. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 97–108, 1999.

[9] 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.

[10] 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.

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

[12] C. Calero, M. Piattini, and M. Genero. Empirical validation of referential integrity metrics. *Information & Software Technology*, 43(15):949–957, 2001.

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

[14] 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.

[15] S.J. Cammarata, P. Ramachandra, and D. Shane. Extending a relational database with deferred referential integrity checking and intelligent joins. In *ACM SIGMOD Conference*, pages 88–97, 1989.

[16] S. Chaudhuri. An overview of query optimization in relational systems. In *ACM PODS Conference*, pages 84–93, 1998.

[17] 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.

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

[19] 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.

[20] E.F. Codd. Extending the database relational model to capture more meaning. *ACM TODS*, 4(4):397–434, 1979.

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

[22] M. Costa and H. Madeira. Handling big dimensions in distributed data warehouses using the dws technique. In *DOLAP '04: Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, pages 31–37, 2004.

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

[24] 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.

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

[26] J.W. Fisher and D.J. Berndt. Creating false memories: Temporal reconstruction errors in data warehouses. In *Eleventh Workshop on Technologies and Systems*, New Orleans, 2001.

[27] C. Francalanci and B. Pernici. Data quality assessment from the user's perspective. In *IQIS '04: Proceedings of the 2004 international workshop on Information quality in information systems*, pages 68–73, 2004.

[28] 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.

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

[30] R. Gallersdörfer and M. Nicola. Improving performance in replicated databases through relaxed coherency. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*, pages 445–456, 1995.

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

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

[33] 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.

[34] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 1st edition, 2001.

[35] B.M. Horowitz. A run-time execution model for referential integrity maintenance. In *IEEE ICDE Conference*, pages 548–556, 1992.

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

[37] T. Johnson, A. Marathe, and T. Dasu. Database exploration and Bellman. *IEEE Data Engineering Bulletin*, 26(3):34–39, 2003.

[38] W. Kim. On optimizing an sql-like nested query. *ACM Trans. Database Syst.*, 7(3):443–469, 1982.

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

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

[41] 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.

[42] E. Lau and S. Madden. An integrated approach to recovery and high availability in an updatable, distributed data warehouse. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 703–714, 2006.

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

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

[45] E.P. Lim and R.H. Chiang. The integration of relationship instances from heterogeneous databases. *Decis. Support Syst.*, 29-2(3-4):153–167, 2000.

[46] V.M. Markowitz. Safe referential structures in relational databases. In *VLDB*, pages 123–132, 1991.

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

[48] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. Technical report, Ithaca, NY, USA, 2000.

[49] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory*, 49(9):2213– 2218, Sept. 2003.

[50] A. Motro and I. Rakov. Estimating the quality of data in relational databases. In *IQ*, pages 94–10, 1996.

[51] D. Mukhopadhyay and G. Thomas. Practical approaches to maintaining referential integrity in multidatabase systems. In *RIDE-IMS*, pages 42–49, July 1993.

[52] 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.

[53] 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.

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

[55] C. Ordonez, J. García-García, and Z. Chen. Measuring referential integrity in distributed databases. In *ACM First Workshop on CyberInfrastructure: Information Management in eScience, CIMS*, pages 61–66, 2007.

[56] M.T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 2nd edition, 1999.

[57] E. Pacitti, P. Minet, and E. Simon. Replica consistency in lazy master replicated databases. *Distrib. Parallel Databases*, 9(3):237–267, 2001.

[58] E. Pacitti and E. Simon. Update propagation strategies to improve freshness in lazy master replicated databases. *The VLDB Journal*, 8(3-4):305–318, 2000.

[59] L. Pipino, Y.W. Lee, and R.Y. Wang. Data quality assessment. *ACM CACM*, 45(4):211–218, 2002.

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

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

[62] M. Scannapieco and C. Batini. Completeness in the relational model: A comprehensive framework. In *IQ Conference*, 2004.

[63] M. Serrano, C. Calero, J. Trujillo, S. Lujan-Mora, and M. Piattini. Empirical validation of metrics for conceptual models of data warehouses. In *CAISE*, pages 506–520, 2004.

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

[65] C. Türker and M. Gertz. Semantic integrity support in SQL: 1999 and commercial (object-)relational database management systems. *VLDBJ*, 10(4):241–269, 2001.

[66] R.Y. Wang, M.P. Reddy, and H.B. Kon. Toward quality data: an attribute-based approach. *Decis. Support Syst.*, 13(3-4):349–372, 1995.

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

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

[69] H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *OSDI'00: Proceedings of the 4th conference on Symposium on Operating System Design & Implementation*, pages 21–21, 2000.

[70] H. Yu and A. Vahdat. The costs and limits of availability for replicated services. *ACM Trans. Comput. Syst.*, 24(1):70–113, 2006.

[71] 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.