Static Analysis of Partial Referential Integrity for Better Quality SQL Data

Sebastian Link  
Department of Computer Science  
University Of Auckland  
s.link@auckland.ac.nz

Mozhgan Memari  
Department of Computer Science  
University Of Auckland  
mmem525@aucklanduni.ac.nz

ABSTRACT

Referential integrity ensures the consistency of data between database relations. The SQL standard proposes different semantics to deal with partial information under referential integrity. Simple semantics neglects tuples with nulls, and enjoys built-in support by commercial database systems. Partial semantics does check tuples with nulls, but does not enjoy built-in support. We investigate this mismatch between the SQL standard and real database systems. Indeed, insight is gained into the trade-off between cleaner data under partial semantics and the efficiency of checking simple semantics. The cost for referential integrity checking is evaluated for various dataset sizes, indexing structures and degrees of cleanliness. While the cost of partial semantics exceeds that of simple semantics, their performance trends follow similar patterns under growing database sizes. Applying multiple index structures and exploiting appropriate validation mechanisms increase the efficiency of checking partial semantics.

Keywords

Data quality, Efficiency, Index, Partial referential integrity, Performance analysis, SQL standard.

INTRODUCTION

After the introduction of the relational model of data in the early 1970s (Codd, 1970), the data definition and query language, SQL, became the industry standard for data management. However, there are features that distinguish SQL from the relational model (Hartmann and Link, 2012). For instance, SQL permits null markers to make data acquisition and processing more efficient in practice. The implications of this feature are not well understood yet. In particular, there are data quality issues associated with incomplete data (Fan and Geerts, 2012; Müller and Freytag, 2005). Lack of completeness makes data in the database dirty which might have fatal consequences in the real world (Fan et al., 2012). As an example, “the Data Warehousing Institute (TDWI) estimates that poor quality customer data costs U.S. businesses a staggering 611 billion a year in postage, printing, and staff overhead” (Eckerson, 2002). These costs explain the increasing demand for data quality tools. By exploiting inference and reasoning, integrity constraints constitute a principled approach for detecting inconsistency and improving data quality (Fan et al., 2012; Grefen and Apers, 1993). Here we focus on referential integrity constraints.

Referential integrity checks the validity of references between data (Codd, 1970). For higher data quality, the SQL standard recommends several different semantics, including simple and partial. Under simple semantics, referencing tuples with null marker occurrences on foreign key attributes never violate referential integrity, by default. Under partial semantics, every referencing tuple with null marker occurrences on their foreign key attributes requires some tuple in the referenced relation with values that match all the corresponding non-null values of the referencing tuple. Therefore, simple and partial semantics are different whenever a null marker occurs on some foreign key attribute. Thus, partial integrity ensures a higher degree of integrity in partial relations (Bravo and Bertossi, 2006; Härder and Reinert, 1996).

Example 1 shows the relation schemata “Medicare” with the primary key {Card-ID, Name}, and “Diagnosis” with the foreign key referencing the Medicare primary key. The two tables show data related to a Medicare cards system (Australian Medicare, 2012). Each Card-ID can be linked to different people in a family who live at the same address. The Diagnosis table shows the information from diagnostic test types for people linked to their Medicare card information. Referential integrity checks that each foreign key value in the Diagnosis table has a matching primary key value in the Medicare table. Under simple semantics the only violation of referential integrity occurs on the first tuple of the Diagnosis table. Partial semantics is also violated by the second tuple of the Diagnosis table.
Example 1 illustrates the potential benefit of partial integrity on data quality. Indeed, as the second tuple in the Diagnosis table indicates, violations of partial integrity suggest that the Null marker occurrence should be interpreted as non-existent. Similarly, as the third tuple in the Diagnosis table indicates, satisfaction of partial integrity suggests that the Null marker occurrence should be interpreted as “values exists, but currently unknown”. Here, possible values that the Null marker suggests are “John Harris” or “Nina Harris”.

Unfortunately, to our knowledge, none of the commercial and open source database systems provide built-in support for partial semantics (Härder et al., 1996; Türker and Gertz, 2001). In other words, users must implement partial semantics themselves in order to take advantage of the benefits illustrated above. Based on previous research (Härder et al., 1996; Grefen et al., 1993; Türker et al., 2001; Zhang, Kuan Tan, Zhang, Lin, Wang, Zhang and Mei, 2011), the validation of partial semantics is a highly intricate matter.

This gap between the SQL standard and the actual implementation in relational database systems must be addressed. Therefore, the trade-off between qualitatively better data based on partial semantics and the arising costs must be understood in detail. Our main technical contribution is the identification and comparison of proper mechanisms that validate partial referential integrity. We identify the penalty of partial referential integrity checking in static databases, such as data warehouses. We explore different factors, including different queries and mechanisms for validating referential integrity, and various data structures with different levels of cleanliness. A variety of experiments determine the cost of detecting referential integrity violations in databases. In particular, we compare the performance of checking simple and partial semantics to identify the costs for achieving better quality SQL data. Our analysis suggests that partial referential integrity can still be validated efficiently and follows patterns that are more costly, but similar to the validation of simple semantics. Developers can use our techniques to make more informed decisions about the feasibility of validating partial referential integrity in their domain of interest.

**Organization.** Preliminary definitions and related work are discussed in the next section. Experiment planning and the variables in our tests are described in the third section. Then we summarize our experiments, the results and the data analysis. We conclude in the last section and present an overview of future work.

**PRELIMINARY DEFINITIONS AND RELATED WORK**

In this section we further explain how partial semantics can improve the quality and consistency of data in the databases. We comment on related work, and explain our empirical study in identifying the costs of applying partial semantics.

**Referential Integrity**

Referential constraints constitute a fundamental concept that targets higher quality data in databases. Referential integrity maintains the relationship between a referencing relation C called CHILD table and the referenced relation P named PARENT table. This constraint holds between a sequence of attributes \( (f_1, f_2, \ldots, f_n) \) called foreign key of \( C \) and a sequence of attributes \( (k_1, k_2, \ldots, k_n) \) as a candidate key of \( P \). For \( i = 1, \ldots, n \), the domains of \( f_i \) and \( k_i \) must match. The foreign key values of the CHILD table are allowed to have null marker occurrences. Referential integrity checks whether for each tuple \( c \) in \( C \) there is some

---

### Table 1: Medicare table

<table>
<thead>
<tr>
<th>Card-ID</th>
<th>Name</th>
<th>Address</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>John Smith</td>
<td>11 Water St.</td>
<td>45</td>
</tr>
<tr>
<td>23456</td>
<td>John Harris</td>
<td>8 Melrose Rd</td>
<td>35</td>
</tr>
<tr>
<td>23456</td>
<td>Nina Harris</td>
<td>8 Melrose Rd</td>
<td>4</td>
</tr>
</tbody>
</table>

### Table 2: Diagnosis table

<table>
<thead>
<tr>
<th>Result</th>
<th>Date</th>
<th>Card-ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBG</td>
<td>12/9</td>
<td>12345</td>
<td>Helen Smith</td>
</tr>
<tr>
<td>OGTT</td>
<td>12/9</td>
<td>Null</td>
<td>Jack Johnson</td>
</tr>
<tr>
<td>A1c</td>
<td>4/3</td>
<td>23456</td>
<td>Null</td>
</tr>
</tbody>
</table>
tuple in \( P \) such that \( c[f_i] = p[k_i] \) for \( i = 1, \ldots, n \). SQL proposes different semantics to handle null markers on foreign key attributes (Härder et al., 1996; Türker et al., 2001).

**Match Clause**

The standard proposes three options for checking the referential integrity in databases. In fact, the “Match clause” options specify the degree of the required match between a composite foreign key and a candidate key. According to the selected option, the following conditions are applied for checking referential integrity.

- Under “Simple” match, referential integrity is satisfied if for each row \( c \in C \), either all the referencing attributes \( f_i \) have matching values in the corresponding attributes \( k_i \) for some tuple \( p \in P \), or at least one value of the referencing attributes is null. This means:

\[
\forall c \in C : \left( \bigwedge_{i=1}^{n} c.f_i \neq \text{null} \right) \Rightarrow \exists p \in P : \left( \bigwedge_{i=1}^{n} c.f_i = p.k_i \right)
\]

Therefore, under simple semantics, referential integrity is never violated by \( \text{CHIL} \)D tuples with null marker occurrences on their foreign keys.

- Under “Partial” match, referential integrity is satisfied if for each row \( c \in C \), either at least one attribute \( f_i \) is null and the rest of not null attributes match the referenced attributes in tuple \( p \in P \), or all the foreign key attributes in \( c \) are not null and have matching values on \( k_i \) in \( p \). More formally:

\[
\forall c \in C : \left( \bigwedge_{i=1}^{n} c.f_i \neq \text{null} \right) \Rightarrow \exists p \in P : \left( \bigwedge_{i=1}^{n} c.f_i = \text{null} \bigvee c.f_i = p.k_i \right)
\]

In contrast to simple semantics, partial semantics also requires partially defined child tuples to respect referential integrity. Note that every database which satisfies partial referential integrity also satisfies simple referential integrity, but not vice versa. Therefore, one might say that a database which satisfies simple referential integrity but not partial referential integrity is of less quality than a database that satisfies partial referential integrity (Bravo et al., 2006; Härder et al., 1996; Türker et al., 2001).

- In this paper we do not consider the third option, namely “Full” match.

**Related Work**

Referential integrity was introduced by Codd in the 1970s (Codd, 1970). Since then it has been studied as a tool to control the semantic correctness of the data in databases (Fan et al., 2012; Grefen et al., 1993; Hartmann, Leck and Link, 2011; Hartmann and Link, 2012; Müller et al., 2005).

Methods, requirements and specifications for implementing referential integrity in database management systems have been proposed (Ordonez et al., 2008; Zhang et al., 2011), along with optimizing existing mechanisms (Grefen et al., 1993; Ordonez et al., 2008; Türker et al., 2001). However, none of these methods covers all the features in regard to referential integrity checking (Grefen et al., 1993; Türker et al., 2001; Zhang et al., 2011).

For improving data quality in the presence of null markers, the SQL standard introduced more sophisticated ways to enforce referential integrity in 1992. Härder et al. studied different features of the match clause without implementation and experimental considerations. Based on their study, considerable numbers and kinds of queries (Härder et al., 1996) are required to enforce partial referential integrity. They highlighted the need for further studies to indicate the entire costs of referential integrity at the system level (Härder et al., 1996). Some research, like (Bravo et al., 2006; Ordonez et al., 2008; Tseng, Chen and Yang, 1993; Türker et al., 2001), suggest to apply data semantics in constraint checking, and to repair query answers. In this work, partial semantics has been explained only briefly or was only considered as future work.

In summary, previous research has highlighted the benefits of enforcing partial semantics and given some theoretical estimates of its complexity, but no algorithmic and experimental studies were conducted that show the costs involved.
Our work

The goal of this research is to determine whether the “cost of referential integrity maintenance” (Härder et al., 1996, p 18) outweights its usefulness in terms of better quality SQL data. In addressing the goal, we have conducted performance tests on synthetic databases in MySQL. Firstly, we investigate how semantics and queries for checking referential integrity affect the performance. We examine the difference in performance for partial and simple semantics as a scale for illuminating the feasibility of exploiting partial semantics in static databases. Furthermore, the impact of different levels of cleanliness and different index structures on the performance are studied.

EXPERIMENT PLANNING

Our experiments were conducted on a machine with Core i7 with the latest Windows 7 Home Premium Edition, employing embedded procedures for the MySQL relational database system using Microsoft Visual C++.

The study examines the influence of four independent variables on the time of referential integrity checking: queries and mechanisms, size of the datasets, the degree of the cleanliness of the foreign key, and indexing structures. We aim to understand how these variables affect the referential integrity validation time in a data warehouse.

Our experiments used one PARENT and one CHILD table, with a 2-attribute \((k_1, k_2)\) key and a 2-attribute \((f_1, f_2)\) foreign key relationship. To populate these tables, we generated synthetic datasets of various sizes ranging from 1000 to 1000k tuples in the PARENT table uniquely identified by the \((k_1, k_2)\) key. The size of the CHILD table is always 1.5 times bigger than that of the PARENT table. The foreign key in the CHILD table features null marker occurrences. Tuples in the CHILD table fall into different categories: those which are not null on any foreign key attribute, those which are null on exactly one attribute, and those which are null on both.

Mechanisms for Referential Integrity

Different mechanisms are exploited for checking referential integrity in the experiments. These mechanisms check the relationship between the key in PARENT table and the foreign key and bookmark any occurring violations. We compare the performance of these mechanisms to determine their efficiency in checking referential integrity in warehouse databases.

The three mechanisms for checking referential integrity have the following SQL statements. QUERY 1 and QUERY 2 are queries for partial semantics and QUERY 3 is for simple semantics. Referential integrity is satisfied if the queries return 0, that is, no violations occur. Execution times show their performance for checking referential integrity.

- **QUERY 1** for partial semantics:
  
  $$\text{SELECT COUNT (*) FROM CHILD WHERE NOT EXISTS (SELECT * FROM PARENT WHERE ((PARENT.} k_1=\text{CHILD.} f_1 \text{ AND PARENT.} k_2=\text{CHILD.} f_2) \text{ OR (PARENT.} k_1=\text{CHILD.} f_1 \text{ AND CHILD.} f_2 \text{ IS NULL}) \text{ OR (CHILD.} f_1 \text{ IS NULL AND CHILD.} f_2 \text{ IS NULL}) \text{ OR (CHILD.} f_1 \text{ IS NULL AND PARENT.} k_2=\text{CHILD.} f_2));}$$

- **QUERY 2** for partial semantics:
  
  $$\text{SELECT COUNT (*) FROM CHILD WHERE NOT EXISTS (SELECT * FROM PARENT WHERE (CHILD.} f_1 \text{ IS NOT NULL AND CHILD.} f_2 \text{ IS NOT NULL AND PARENT.} k_1=\text{CHILD.} f_1 \text{ AND PARENT.} k_2=\text{CHILD.} f_2) \text{ UNION (SELECT * FROM PARENT WHERE (CHILD.} f_1 \text{ IS NOT NULL AND PARENT.} k_1=\text{CHILD.} f_1 \text{ AND CHILD.} f_2 \text{ IS NULL}) \text{ UNION (SELECT * FROM PARENT WHERE (CHILD.} f_1 \text{ IS NULL AND PARENT.} k_2=\text{CHILD.} f_2 \text{ AND CHILD.} f_2 \text{ IS NOT NULL}) \text{ UNION (SELECT * FROM PARENT WHERE (CHILD.} f_1 \text{ IS NOT NULL AND CHILD.} f_2 \text{ IS NULL));}}$$

- **QUERY 3** for simple semantics:
  
  $$\text{SELECT COUNT (*) FROM CHILD WHERE (CHILD.} f_1 \text{ IS NOT NULL AND CHILD.} f_2 \text{ IS NOT NULL AND NOT EXISTS (SELECT * FROM PARENT WHERE (PARENT.} k_1=\text{CHILD.} f_1 \text{ AND PARENT.} k_2=\text{CHILD.} f_2));}$$

- Built-In foreign key constraints are also used to check simple semantics, and their execution time is compared with the previous three mechanisms.
  
  **ALTER TABLE CHILD ADD CONSTRAINT FOREIGN KEY Foreignkey (f_1, f_2) REFERENCES PARENT (k_1, k_2);**

For benchmarking the performance of the BUILT-IN FOREIGN KEY CONSTRAINT against QUERY 1 and QUERY 2 we horizontally decompose the CHILD table into three tables Fk1, Fk2, and Fk3. Then we apply the BUILT-IN FOREIGN KEY CONSTRAINT for checking simple referential integrity on the three tables. The sum of the three execution times is then the time of checking referential integrity in the CHILD table by partial semantics. This mechanism is named METHOD 1.
Decomposing the CHILD table into Fk1, Fk2 and Fk3 tables:

\[
Fk1 = \sigma_{\varphi_1 \varphi_2} (\text{CHILD}) \text{ WHERE } \varphi_1 = (c.f_1 \neq \text{Null}) \land (c.f_2 \neq \text{Null}) \\
\text{AND } \varphi_2 = (c.f_1 = \text{Null}) \lor (c.f_2 = \text{Null}) \\
Fk2 = \sigma_{(c.f_1 \neq \text{Null}) \land (c.f_2 = \text{Null})} (\text{CHILD}) \\
Fk3 = \sigma_{(c.f_1 = \text{Null}) \land (c.f_2 \neq \text{Null})} (\text{CHILD})
\]

**METHOD 1:**

ALTER TABLE Fk1 ADD CONSTRAINT FOREIGN KEY Foreignkey (f1, f2) REFERENCES PARENT (k1, k2);
ALTER TABLE Fk2 ADD CONSTRAINT FOREIGN KEY Foreignkey (f1) REFERENCES PARENT (k1);
ALTER TABLE Fk3 ADD CONSTRAINT FOREIGN KEY Foreignkey (f2) REFERENCES PARENT (k2);

Query 3 has also been exploited for checking simple referential integrity on the decomposed tables. The execution times for checking these three tables show the time of checking referential integrity in the CHILD table by partial semantics. We refer to this mechanism by METHOD 2.

**METHOD 2:**

SELECT COUNT (*) FROM Fk1 WHERE ((Fk1. f1 IS NOT NULL AND Fk1. f2 IS NOT NULL) AND NOT EXISTS (SELECT * FROM PARENT WHERE (PARENT. k1 = Fk1. f1 AND PARENT. k2 = Fk1. f2)));
SELECT COUNT(*) FROM Fk2
WHERE NOT EXISTS (SELECT * FROM PARENT WHERE (PARENT. k1 = Fk2. f1));
SELECT COUNT(*) FROM Fk3
WHERE NOT EXISTS (SELECT * FROM PARENT WHERE (PARENT. k2 = Fk3. f2));

**Cleanliness of the CHILD Table**

For the experiments we considered 20%, 50%, and 80% of all n tuples to be clean. The distribution of null marker occurrences between the two foreign key attributes is equal for all three degrees. Table 3 shows a summary of scenarios based on the distribution of null markers in the data samples with n tuples.

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution of Null in the foreign key</td>
<td>20%</td>
<td>50%</td>
</tr>
<tr>
<td>Tuples with null only on 1\textsuperscript{st} attribute</td>
<td>n\times0.2\times0.45</td>
<td>n\times0.5\times0.45</td>
</tr>
<tr>
<td>Tuples with null only on 2\textsuperscript{nd} attribute</td>
<td>n\times0.2\times0.45</td>
<td>n\times0.5\times0.45</td>
</tr>
<tr>
<td>Tuples with null on both attribute</td>
<td>n\times0.2\times0.1</td>
<td>n\times0.5\times0.1</td>
</tr>
</tbody>
</table>

Table 3: Summary of scenarios for the distribution of null markers in the foreign key

**Index Structures In Storage Engines**

The experiments have been conducted on the MyISAM and InnoDB storage engines in MySQL. InnoDB supports B-Tree and Hash indexes. MyISAM only supports B-Tree indexing and does also not support internal foreign key constraints.

In our tests we utilize two composite indexes over (P.k\textsubscript{1}, P.k\textsubscript{2}) and (P.k\textsubscript{3}, P.k\textsubscript{1}), respectively, on the PARENT table. Two composite index structures (C.f\textsubscript{1}, C.f\textsubscript{2}) and (C.f\textsubscript{2}, C.f\textsubscript{1}) are also exploited on the foreign key in the CHILD table. (Härder et al., 1996, p 1) suggested to apply “a combination of multiple ...” index structures to enforce partial semantics. Hence, we examine whether the index structures over (P.k\textsubscript{3}, P.k\textsubscript{1}) and (C.f\textsubscript{2}, C.f\textsubscript{1}) improve the performance of partial referential integrity.
checking. We also run experiments by applying one index over \((P.k_1, P.k_2)\) and one over \((C.f_1, C.f_2)\). Table 4 summarizes the index structures that have been used for our experiments.

<table>
<thead>
<tr>
<th>Index Code</th>
<th>Storage Engine</th>
<th>Index structure on Child table</th>
<th>Index structure on Parent table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>InnoDB</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>2</td>
<td>InnoDB</td>
<td>B-Tree over ((C.f_1, C.f_2)) and ((C.f_2, C.f_1))</td>
<td>B-Tree over ((P.k_1, P.k_2)) and ((P.k_2, P.k_1))</td>
</tr>
<tr>
<td>3</td>
<td>InnoDB</td>
<td>B-Hash over ((C.f_1, C.f_2)) and ((C.f_2, C.f_1))</td>
<td>B-Hash over ((P.k_1, P.k_2)) and ((P.k_2, P.k_1))</td>
</tr>
<tr>
<td>4</td>
<td>MyISAM</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>5</td>
<td>MyISAM</td>
<td>B-Tree over ((C.f_1, C.f_2)) and ((C.f_2, C.f_1))</td>
<td>B-Tree over ((P.k_1, P.k_2)) and ((P.k_2, P.k_1))</td>
</tr>
<tr>
<td>6</td>
<td>InnoDB</td>
<td>B-Tree over ((C.f_1, C.f_2))</td>
<td>B-Tree over ((P.k_1, P.k_2))</td>
</tr>
<tr>
<td>7</td>
<td>MyISAM</td>
<td>B-Tree over ((C.f_1, C.f_2))</td>
<td>B-Tree over ((P.k_1, P.k_2))</td>
</tr>
</tbody>
</table>

Table 4: Summary of the index structures in the tests

EXPERIMENTAL RESULTS AND DATA ANALYSIS

We report on the results of the experiments, exploring how each variable affects the performance of referential integrity checking. In total, the experiments include 1270 tests to evaluate partial semantics performance and 568 tests for analysing the simple semantics performance.

Impact of the Mechanisms to Check Referential Integrity

We compare the performances of different mechanisms under different semantics in this section.

Figure 1 illustrates the results of the tests conducted on two existing datasets using a B-tree index on InnoDB, which is Index-Code 2 shown in Table 4. Table 5 provides detailed information about the execution times when there are 80% null values in the foreign key attributes.

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of tests</th>
<th>Execution Times in seconds:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mean</td>
</tr>
<tr>
<td>QUERY 1</td>
<td>16</td>
<td>700.08</td>
</tr>
<tr>
<td>QUERY 2</td>
<td>25</td>
<td>55.69</td>
</tr>
<tr>
<td>METHOD 1</td>
<td>25</td>
<td>0.839</td>
</tr>
<tr>
<td>METHOD 2</td>
<td>25</td>
<td>4.103</td>
</tr>
<tr>
<td>QUERY 3</td>
<td>25</td>
<td>0.288</td>
</tr>
<tr>
<td>Built-In Foreign key Constraint</td>
<td>25</td>
<td>7.097</td>
</tr>
</tbody>
</table>

Table 5: Partial and simple semantics' performances, Index Btree, InnoDB storage engine, 20% Cleanliness level

According to Table 5, the time of partial referential integrity checking for the mentioned index structures and degrees of cleanliness, is from an average of 0.839s (Min=0 for 1000 rows and Max=3.97s for 1000k tuples in Parent table), when Method 1 is applied, to 700.08s (Min=0.219s for 1000 rows and Max=2330.4s for 1000k rows) for Query 1. Figure 1 also
shows that the execution time of QUERY 1 rises exponentially with the dataset size. Mechanisms, different from QUERY 1, show a linear correlation between the time and the dataset size.

![Graph showing partial RI results, for B-tree Index on InnoDB](image)

**Figure 1: The Partial RI results, for B-tree Index on InnoDB**

Next we exploited the query optimization in MySQL to compare the performance of QUERY 1 and QUERY 2. Based on the results of “EXTENDED EXPLAIN” for the experiment with 100k tuples in the PARENT table, for example, QUERY 1 reads 150,245 tuples on the main “SELECT” statement in the query and 100,332 tuples in the dependent sub-query. QUERY 2 checks 150,245 tuples in the main “SELECT” statement and only 28,770 tuples in the dependent queries and “UNION” statement. That is, QUERY 1 performs a full table scan without optimization. The Mann-Whitney test reveals that the differences between QUERY 1 and the other mechanisms are significant at the level of 0.05 (Table 6). Because of these significant differences, we do not consider the results of this query for the rest of the data analysis here.

<table>
<thead>
<tr>
<th>Compares to:</th>
<th>U</th>
<th>R</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY 2</td>
<td>10704.5</td>
<td>-0.226</td>
<td>0.048*</td>
</tr>
<tr>
<td>METHOD 1</td>
<td>6153.0</td>
<td>-0.614</td>
<td>0.000*</td>
</tr>
<tr>
<td>METHOD 2</td>
<td>7565.50</td>
<td>-0.403</td>
<td>0.000*</td>
</tr>
</tbody>
</table>

* The mean difference is significant at the 0.05 level.

**Table 6: Comparing the performances of QUERY 1 and other partial semantics mechanisms**

QUERY 2 shows better performance than QUERY 1. However, there is still a significant difference between the execution times of this and other methods. The average time taken by QUERY 2 is 55.69s (Min=0.031 for 1000 rows and Max=301.94s...
for 1000k tuples in PARENT table). There is a linear correlation between the dataset size and the execution time for this query and its figure bears resemblance to the simple semantics’ pattern. Since a decomposition of the CHILD table is not feasible for developers, QUERY 2 represents an alternative mechanism for applying partial referential integrity. Table 5 and Figure 1 also show that METHOD 1 provides the best performance for partial semantics.

The performance of simple semantics mechanisms have also been illustrated in Figure 2. The best option for the simple semantics is QUERY 3. Contrary to expectations, checking simple referential integrity by applying built-in foreign key constraints does not provide better performance than QUERY 3. The reason is that internal mechanisms were developed for the dynamic task of updating the database. This finding provides strong motivation for developing a mechanism for checking referential integrity for static databases, even under simple semantics.

![Figure 2: Execution times to check simple referential integrity](image)

Although the results illustrate that the time taken for partial semantics is always more than simple semantics, the figures seem to follow the same pattern\(^1\). This suggests the feasibility of checking partial referential integrity in data warehouses, in MySQL.

**Exploiting Index Structures**

The aim is to assess the impact of index structures on the performance of checking referential integrity (Härder et al., 1996). Our focus is the impact of applying different indices and multiple index levels on the performance at the system level.

Firstly, we compare the performance between two sets of experiments. For the first set of tests we check referential integrity between the PARENT table with no primary key or index level over P.k\(_1\) or/and P.k\(_2\) and the CHILD table with no defined index over C.f\(_1\) or/and C.f\(_2\). The second set of tests, however, validates referential integrity when one of the Index-Codes 2, 3 or 5, shown in Table 4, is used on the PARENT and CHILD tables. The execution time for the first set is significantly greater than the time of referential integrity checking using index structures. Using the Mann-Whitney test (Field, 2009), Table 7 shows the difference in time for checking referential integrity between these two sets of tests for each mechanism. All the differences are significant at the 0.005 level, except for QUERY 1 which is not significant even on the 95% confidence level. This supports our previous finding from the last section. Indeed, the time of checking partial referential integrity by QUERY 2

\(^1\) Except for QUERY 1.
increases exponentially when no index structure is used. However, the results show a linear correlation between this query's execution time and the size of the datasets for Index-Codes 2, 3 and 5.

Another unanticipated finding of this study is the performance of QUERY 2 for Index-Codes 6 and 7, which use indices \((P.k_i, P.k_j)\) and \((C.f_i, C.f_j)\) on PARENT and CHILD tables, respectively. The results show an exponential increase in the size of the datasets. The time for checking partial referential integrity with Index-Codes 6 or 7 is also significantly greater than the results of the tests for Index-Codes 2, 3 or 5, at the 0.05 level, although the performance is considerably better than the tests with no designed indices. The results show that the indices only optimize the search through tuples with null in the second foreign key attribute. For checking the tuples with null in the first attribute, the entire table is scanned. This confirms our assumption based on (Härder et al., 1996) that applying multiple indices improves the performance of partial semantics.

<table>
<thead>
<tr>
<th>For the Query</th>
<th>U</th>
<th>R</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY 1</td>
<td>9271.00</td>
<td>-0.377</td>
<td>0.109</td>
</tr>
<tr>
<td>QUERY 2</td>
<td>5495.00</td>
<td>-0.545</td>
<td>0.000*</td>
</tr>
<tr>
<td>METHOD 1</td>
<td>3758.50</td>
<td>-0.634</td>
<td>0.000*</td>
</tr>
<tr>
<td>METHOD 2</td>
<td>722.00</td>
<td>-0.701</td>
<td>0.000*</td>
</tr>
<tr>
<td>QUERY 3</td>
<td>4510.00</td>
<td>-0.594</td>
<td>0.000*</td>
</tr>
<tr>
<td>BUILT-IN FOREIGN KEY</td>
<td>913.00</td>
<td>-0.666</td>
<td>0.000*</td>
</tr>
</tbody>
</table>

*. The mean difference is significant at the 0.005 level.

Table 7: Difference of performances under index structures (Index-codes 2, 3 and 5) vs. no index structures

Applying the Kruskal-Wallis test (Field, 2009) shows no significant difference in the benefits gained from defining B-tree indices or Hash indices in InnoDB. Also we did not find significant differences in the performance of the test runs on InnoDB and MyISAM using B-tree indices.

Impact of Cleanliness

According to the results illustrated in Figure 1 to 3, the performance decreases when there are more nulls on the foreign key attributes. Using the Kruskal-wallis test, the difference between the levels of cleanliness for partial semantics is significant, \(H(2)=7.94\) and \(P=0.019<0.05\).

We used the Mann-Whitney test to explain the differences in performance between two levels of cleanliness. The differences between 80% and 50%, and also between 80% and 20% nulls on foreign key attributes are significant, with \(p\)-value 0.004 and 0.096, respectively. Contrary to our expectations, this finding shows that an increase in null values increases efficiency.

CONCLUSION AND FUTURE WORK

This study investigated the cost of partial referential integrity checking within static databases. We examined the difference in performance under partial and simple semantics, the impact of indices, and the usefulness of partial semantics. Our results indicate that the performance is considerably dependent on the mechanisms used for checking. We found the performance of various user-defined queries to be considerably different. Query optimization does not improve the performance in all cases. This highlights the necessity of providing built-in support for partial referential integrity within database management systems. Optimizations of partial referential integrity mechanisms result in higher quality data at lower cost. Our results show that built-in foreign key constraints, implemented for maintaining simple referential integrity in update operations, do not provide the best performance within a static database. In fact, our query based on simple semantics was more efficient. Again this highlights the necessity of implementing an appropriate mechanism for referential integrity checking, in particular for data warehouses. Our experiments confirm previous theoretical findings that suitable multiple indices improve the performance of partial semantics. Indices have a significant impact on the performance of simple semantics as well. Surprisingly, we found that having more null values in the foreign key attributes decreases the time for checking partial semantics. Finally, as an important implication, this study showed that it is indeed more costly to apply partial semantics in referential integrity maintenance, but the cost for the simple and partial semantics seem to scale in the same way. This finding provides an insight into the feasibility of validating partial referential integrity in data warehouses to reduce data...
incompleteness at a reasonable cost. Future research will analyse the performance of dynamic partial referential integrity enforcement when updating the database. Furthermore, we will address the performance of validating partial semantics in other common database management systems. It is important to determine the cost of partial semantics for composite keys with more than two attributes. It is also interesting to analyse how much real world applications benefit from partial semantics along with the feasibility of exploiting this semantics in these applications.

REFERENCES