FusionDB: Conflict Management System for Small-Science Databases

Karim Ibrahim, Nathaniel Selvo, Mohamad El-Rifai, and Mohamed Eltabakh
Computer Science Department, Worcester Polytechnic Institute
Worcester, MA, USA
kaibrahim@cs.wpi.edu, nselvo@cs.wpi.edu, melrifai@cs.wpi.edu,
meltabakh@cs.wpi.edu

ABSTRACT

In this paper, we demonstrate the *FusionDB* system; an extended relational database engine for managing conflicts in small-science databases. In small sciences, groups—each consists of few scientists—may share and exchange parts of their own databases among each other to foster collaboration. The goal of such sharing, especially when done at early stages of the discovery process, is not to build a warehouse or a unified schema, instead the goal is to compare and verify results, detect and assess conflicts, and possibly modify or re-design the discovery process. FusionDB is designed to meet the requirements and address the challenges of such sharing model. We will demonstrate the key functionalities of FusionDB including: (1) Detecting conflicts using a rule-based model over heterogeneous schemas, (2) Assessing conflicts and providing probabilistic estimates for values' correctness, (3) Extended querying capabilities in the presence of conflicts, and (4) Providing curation operations to help scientists resolve and investigate conflicts according to different priorities. FusionDB is realized on top of PostgreSQL DBMS.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data models, Schema and subschema*

Keywords

Conflict management; databases; scientific data.

1. INTRODUCTION

Database management systems play a key role in supporting scientific applications in various domains such as in biology, chemistry, physics, and earth and ocean sciences. In these applications, most discoveries and innovations are not driven by centralized processing, instead, they are fueled by *small sciences* where many small-scale groups of scientists are conducting their own experiments, collecting measurements, and storing their own data in local databases. Then, related groups working on the same (or similar) objects/subjects may collaborate by exchanging and sharing parts of their own data with each other. Such collaboration and sharing of data can be done at early stages of the discovery process with the aim for comparing and verifying results with each other, detecting and assessing conflicts as early as possible, and possibly refining experimental setups or adjusting parameters. These goals are different from those of traditional data integration systems, e.g., [10, 12, 7, 11], that aim for building a unified and consistent warehouse. Thus, the system requirements and desired functionalities in small-science data sharing cannot be fulfilled using existing data integration systems.

Sharing data in small sciences involve several novel and unique data management challenges w.r.t conflict management and resolution for several reasons:

(1) ***Scientific databases inherently contain conflicts that cannot be avoided or eliminated.*** This is because there no single authority for curating the data, different interpretations and observations may lead to different values, and different scientists may have different beliefs about their data. Therefore, trying to create a single consistent instance over the shared data— as in the traditional data integration systems, e.g., [7, 11, 8, 4]— may not be applicable.

(2) ***Availability of conflicting data for analysis and querying.*** Although conflicting data may sometimes lead to confusion, scientists do prefer to have all the data available—even if conflicting—for analysis, querying, and comparison with other values. Therefore, discarding conflicting data as early as possible outside the database system—as in update exchange systems, e.g., [6, 5, 9]— is not desirable as it causes the lose of valuable information before even doing any analysis over the data.

(3) ***Ability to automatically assess conflicts and provide recommendations.*** Scientists are of great need for automatic mechanisms for detecting possible conflicts, assessing them, and probably providing evidence-based recommendations on which values are more likely to be correct (or wrong). Moreover, if a scientist wants to investigate conflicts, then which ones have higher priorities to start with? Certainly, different scientists may have different priorities. Delegating these tasks to end-users–as in current conflict-resolution techniques, e.g., [5, 8, 9]— is usually an overwhelming and error-prone task.

In this paper, we demonstrate *FusionDB* system; an extended relational database management system for supporting data sharing in small sciences. In Figure 1, we illustrate the underlying sharing model where one site S_{host}, e.g., a small scientific lab, may collect a set of databases, denoted by

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.
ACM 978-1-4503-2263-8/13/10.
http://dx.doi.org/10.1145/2505515.2508205.

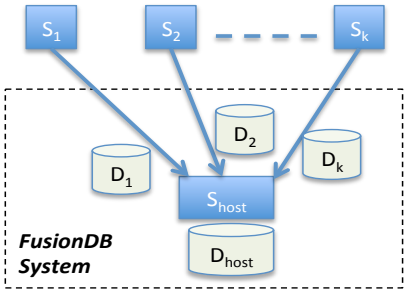

Figure 1: FusionDB Model.

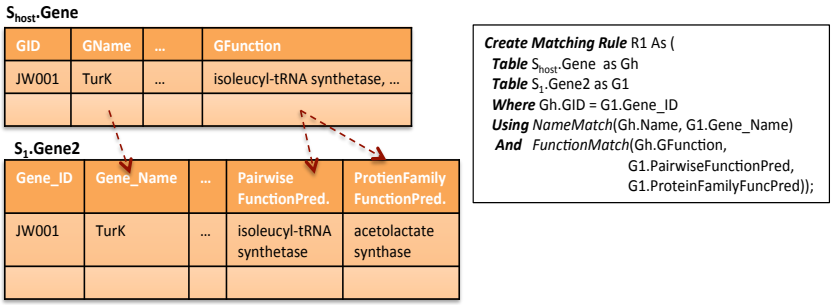

Figure 2: Example of rule-based matching.

$D_1, D_2, ..., D_k$, from several other collaborating sites. Each of these databases has its own structure (schema) and they will all be stored locally at site S_{host} along with S_{host}'s own database D_{host}. Each database D_i provided by site S_i can be only a small subset of the database at that site. In the model, any site can be a host and may receive data from other sites, and in this case each host should run a separate instance of FusionDB system. FusionDB manages the collected datasets and provides the following key novel functionalities:

(1) **Rule-based Conflict Detection:** FusionDB allows the host site, e.g., S_{host} in our example, to define a set of rules, called *matching rules*, for matching records and values. These rules will guide the system to automatically find the records that should compare with each other (entity resolution), how they should compare (comparison mechanism), and how to measure the degree of conflict if exists (quantifying the degree of conflict).

(2) **Conflict Assessment:** If conflicts exist, then a crucial task is to predict and estimate—with a certain degree of confidence— which among the conflicting values are erroneous and which ones are correct. FusionDB provides a conflict assessment mechanism using a probabilistic model and integrates it in query processing.

(3) **Curation Mechanisms for Conflict Resolution:** FusionDB enables scientists to prioritize the existing conflicts for possible resolution or further investigation. We provide different curation operators for different prioritization, e.g., conflicts that affect the largest number of queries are the ones to investigate first, or conflicts with the least degree of mismatch are the ones to resolve first.

(4) **Conflict-aware Query Processing:** FusionDB extends the relational query operators to offer conflict-aware querying capabilities, e.g., users can query only non-conflicting tuples, can specify thresholds to eliminate tuples above a certain degree of conflicts, and can propagate the conflict information along with their queries answers.

2. SYSTEM OVERVIEW & FEATURES

In this section, we briefly highlight the four novel functionalities offered by FusionDB. All what is needed from end-users to utilize the functionalities of FusionDB is to define the *matching rules* introduced in Section 2.1.

2.1 Rule-based Conflict Detection

In order to detect conflicts, the system needs first to know which tuples represent the same object and hence should be compared with each other. Moreover, the system should know how to compare these tuples, e.g., which fields to match and using which functions. These tasks are not straightforward especially under heterogeneous schemas. Automated schema mapping and entity resolution techniques, e.g., [11, 8, 10], usually require user's intervention, at some point, to validate the mappings and correct mismatches. In FusionDB, we deploy a generic and simple user-driven rule-based model to perform the above tasks using the new *Create Matching Rule* command introduced to SQL as follows:

```
Create Matching Rule [<id>] As (
Table D_host.R [As <alias>]
Table D_i.S [As <alias>]
Where Pred(r, s)
Using F1(r.A_i, ..., r.A_j, s.A_k, ..., s.A_m)
[And F2(r.A'_i', ..., r.A'_j', s.A'_k', ..., s.A'_m')
And ...]);
```

Figure 2 illustrates an example of defining a matching rule between two tables storing gene information. In the example, two tuples are comparable to each other if they have the same IDs (the *Where* clause), and to decide whether they are matching or not, the names of the genes are compared using function *NameMatch()*, while the gene functions are compared using function *FunctionMatch()*. Note that fields do not have to be one-to-one match as illustrated in the gene function comparison.

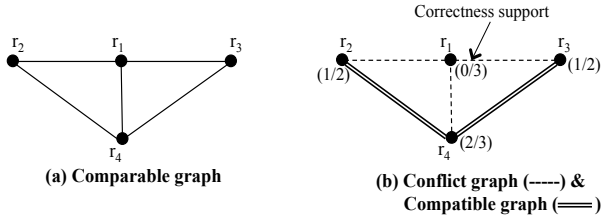

Figure 3: Example of tuple-level *CorrSupp()*.

2.2 Conflict Assessment

FusionDB estimates, with a certain degree of confidence, which among the conflicting values are the erroneous ones and which are the correct ones. This estimation does not only help scientists in resolving conflicts by providing useful hints, but it also enables integrating these estimations in query processing even before resolving the conflicts.

FusionDB computes a degree of confidence, called *correctness support*, at two granularities, tuple-level and attribute-level. The correctness support of a given tuple r is defined as $CorrSupp(r) = r_m/r_n$, where r_m is the number of tuples compatible with r, while r_n is the total number of tuples comparable with r. The example in Figure 3 illustrates the main idea. Assume that we have 4 tuples r_1, r_2, r_3, and r_4 that are comparable to each other w.r.t function $F()$ according to

the graph depicted in Figure 3(a), e.g., r_1 is comparable to the other three tuples while r_2, for example, is comparable to r_1 and r_4 only. Assume that according to the matching rules defined in the system, r_1 is conflicting with the other tuples—as indicated by dashed lines in Figure 3(b)— while the other tuples are compatible—as indicated by double lines in Figure 3(b). From Figure 3(b), we observe that r_1 is not supported by any other tuples, and hence its *CorrSupp(r_1) = 0/3*, while r_4, for example, is supported by 2 tuples out of the 3 comparable tuples, and hence *CorrSupp(r_4) = 2/3*. The other tuples have correctness support of 1/2.

The attribute-level correctness support is defined as $CorrSupp(r.c) = 1 - r_{x_c}/r_n$, where r_{x_c} is the number of tuples conflicting with r because of $r.c$, i.e., there is a function F_i that takes $r.c$ as input and returns a non-zero value, and r_n is as defined before. In contrast to the tuple-level correctness support, the attribute-level support enables the query engine to dynamically compute the correctness support based on only the attributes touched by the query, instead of the entire record (See Section 2.4).

2.3 Curation Mechanisms

FusionDB provides several curation operators that help scientists resolve and investigate conflicts. The database may contain many conflicting tuples, and the question is: *which among these conflicts to investigate first?* FusionDB offers the following operators:

• **ReportConflicts $\Gamma(R)$:** Where given a relation R—which can be an output from a *select* statement— report for each tuple $t \in R$ all conflicting tuples with t. *ReportConflicts* operator reports along with the conflicting tuples, the reasons of the conflict, e.g., which comparison function(s) have failed, and the conflicting score.

• **PrioritizeConflicts $\Psi([\Gamma(R) \mid ALL], sort\_type, direction)$:** Where the first parameter is either a subset of the conflicts that users want to focus on (they are defined using the *ReportConflicts* operator) or all the conflicts in the database (using the *ALL* keyword). The *sort_type* parameter specifies how to sort the conflicts. FusionDB offers three types of sorting criteria: (1) *SCORE*—where conflicts are sorted based on their scores, (2) *CNT*—where conflicting tuples are sorted based on how many other tuples they conflict with, and (3) *POPULARITY*—where conflicting tuples are sorted based on how many queries they appear in. The last parameter in *PrioritizeConflicts* operator specifies the sorting direction, i.e., ascending or descending.

2.4 Conflict-aware Query Processing

FusionDB extends the querying mechanism to seamlessly integrate the conflict information into query processing. Two *"conceptual"* attributes have been added to each tuple in the database, namely *CorreSupp* and *RelCorrSupp*. CorreSupp is of type *double* and represents the tuple-level correctness support, while RelCorrSupp is of type *array[double]* and represents all the attribute-level correctness supports for the give tuple. These attributes are not actually materialized in the database, but can be referenced in the *select* statement like regular attributes. The other extension provided by FusionDB is the automatic propagation of the attribute-level *CorrSupp* values along with the queries's answers, i.e., each attribute value gets annotated with is *CorrSupp* value. For SPJ (select-project-join) queries, each value gets annotated with a single *CorrSupp* value, while in the case of aggregation, duplicate elimination, and set operators, each value may get annotated with an array of *CorrSupp* values resulted from merging multiple identical tuples into one.

3. DEMONSTRATION SCENARIO

We demonstrate the features of FusionDB over a set of publically-available biological databases. More specifically, we will use *Genobase*, *EcoCyc*, *PortEco*, and *ColiBase*. Each dataset has its own schema for storing gene-related information of E.coli. organism. All datasets store information for about 4,100 genes (for a single strain), however, they are not all identical and there are many conflicts in the overlapped information. In the demonstration, we treat one dataset, e.g., *Genebase*, as the host database, and the others will be the sharing-sites databases. And then, we will illustrate, through a Java-enabled GUI, the functionalities presented in Section 2.

4. RELATED WORK

As highlighted in Section 1, the goals and motivation for data sharing in small sciences are unique, which makes state-of-art techniques fall short in managing conflicts in small-science databases. For example, data integration systems, e.g., [7, 8, 12], along with their involved technologies in schema mapping [11, 10, 4], entity resolution [12, 3], ETL (Extract-Transform-Load) techniques [2, 3], and data warehousing [1], aim for creating a single consistent instance of the database and they try to resolve conflicts before building the integrated version of the data. Update-Exchange systems [6, 5, 9] resolve conflicts outside the DBMS and try to keep the database instance at each site consistent. Hence, conflicting data are not available for future analysis or querying.

5. REFERENCES

[1] *Data modeling techniques for data warehousing*. IBM Corp., 1998.

[2] S. Chaudhuri and et. al. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, pages 313–324, 2003.

[3] P. Christen. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *TKDE*, 24(9):1537–1555, 2012.

[4] H. Elmeleegy and et. al. Leveraging query logs for schema mapping generation in U-MAP. In *SIGMOD*, pages 121–132, 2011.

[5] W. Gatterbauer and et. al. Data conflict resolution using trust mappings. In *SIGMOD*, pages 219–230, 2010.

[6] Z. Ives and et. al. The ORCHESTRA Collaborative Data Sharing System. *SIGMOD Rec.*, 37(3):26–32, 2008.

[7] M. Lenzerini. Data integration: A theoretical perspective, 2002.

[8] F. Naumann and M. HŁussler. Declarative Data Merging With Conflict Resolution. In *International Conference on Information Quality*, pages 212–224, 2002.

[9] Y. Qi and K. S. Candan. Ficsr: Feedback-based InConSistency Resolution and query processing on misaligned data sources. In *SIGMOD*, pages 151–162, 2007.

[10] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[11] P. Shvaiko and et. al. A Survey of Schema-Based Matching Approaches. pages 146–171, 2005.

[12] M. Yakout and et. al. Behavior Based Record Linkage. *PVLDB*, 3(1):439–448, 2010.