Indian Journal of Science and Technology, Vol 9(44), DOI: 10.17485/ijst/2016/v9i44/105314, November 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Data Integration - Challenges, Techniques and Future Directions: A Comprehensive Study

Bazeer Ahamed[1*] and T. Ramkumar[2]

[1]Faculty of Computer Science and Engineering, Sathyabama University, Chennai - 600119, Tamil Nadu, India; bazeerahamed@gmail.com

[2]School of Information Technology and Engineering, VIT University, Vellore - 632 014, Tamil Nadu, India; ramooad@yahoo.com

Abstract

Objectives: This paper studies various query reformulation techniques, which are used to convert the intermediate schema to the targeted schema. The techniques such as Ontology based information integration and data integration languages are also reviewed. **Methods/Statistical Analysis**: This paper discusses the techniques used for data integration and also to resolve inconsistencies from the integrated data. Data integration techniques mainly focusing on integration of data in several levels and applying independent or unified query over the data available. **Findings:** Analysis of various techniques done in the paper has led to the identification of several shortcomings and scope for improvements in the available techniques. This identified research directions includes vertical enhancement of wrappers by utilizing a single unified wrapper for all the data sources. Optimizing the queries depending on the data source is also another major requirement to provide efficient and faster results reducing the data retrieval latencies. The paper also advocates other research directions that include identifying duplicates from the retrieved data and performing effective elimination strategies to reduce space consumption. Identifying conflicts and applying strategies to eliminate conflicts is another major area with a huge scope for improvement. **Application/Improvements:** The comprehensive survey also recommends further works in the area of data integration techniques.

Keywords: Conflict Identification, Conflict Resolution, Data Integration, Data Conflicts, Inconsistency Resolution

1. Introduction

Internet is one of the major technologies used in the current era, which provides a huge amount of information not only in readable format, but also downloadable and usable formats. It acts as a huge repository drowning the users with the required information. This vast growth of the internet and the easy access it provides to the masses has been a huge advantage in collection and dispersion of information. Since most of this information is free, it is also being used in many researches. But difficulty arises when the user decides to operate on multiple data sources rather than a single source. If these data sources contain mutually exclusive information then they can be combined and worked on efficiently[1]. But if they contain common information, then problems arise during the process of information retrieval.

Since information fusion is one of the most commonly occurring activities in today's scenario, these discrepancies tend to occur frequently. Methods of dealing with these discrepancies differ in the perspective of the analysis being performed. Most of the structure relates semantics can be solved using the metadata information available, while semantic related conflicts merge as the most difficult to solve scenario[2].

In order to perform effective fusion of the heterogeneous data sources, it becomes mandatory to choose a language that is common to all the data sources[3]. Since such a universal language is not available, the data integration system architecture should be designed with a

*Author for correspondence

custom query language. The efficiency of the query language and the efficiency of the processing components determine the overall efficiency of the system.

This paper describes the architecture of a data integration system and various components involved. It describes the challenges faced by a data integration system and in a broad sense describes the methods that are used to integrate data. It also performs a review of the data integration languages and their association with SQL and presents the research directions.

2. Data Integration Architecture: A Comparison with Traditional DBMS

Data Integration is the process of obtaining and combining multiple data sources for use in an application. The concept of data integration has gained wide prominence after the introduction of the World Wide Web and the huge amount of information associated with it. Though a large amount of data is available, their formats differ considerably due to the unavailability of a regulated schema in the system. Hence data integration becomes mandatory[4].

Query is the basic component of a data integration system. Query is the process of providing constraints to the data integration system to obtain results from it. Though this appears to be a simple process, the inconsistency and heterogeneity of the data in the web makes it a complex process. The choice of query entirely depends upon the innate data integration system that is in place. There is no fixed rule for developing such a language. It depends entirely on the functionality of the base system that is being used. Results of the queries are usually represented as views in the data integration framework. Various broad categories of the data integration system are discussed in section 4. Figure 1 shows the general architecture of a data integration system[5,6].

The query provided by the user is passed to the Query Reformulation phase. The data integration system provides a unified view of various data structures present in the lower level, hence the query being used corresponds to the projected view of data, rather than the specific source format. The Query Reformulation phase enables the conversion of the query into specific formats defined by the base level data sources. A balance is to be maintained in correspondence with the base data stored, when formulating the query. The tradeoff balances the complexity of the query provided to the user and the complexity of the data integration system in place. As the query language becomes more expressive, the design of the data integration system becomes more complex. A traditional DBMS does not have this phase, as all the queries are written for the specific data model being used.

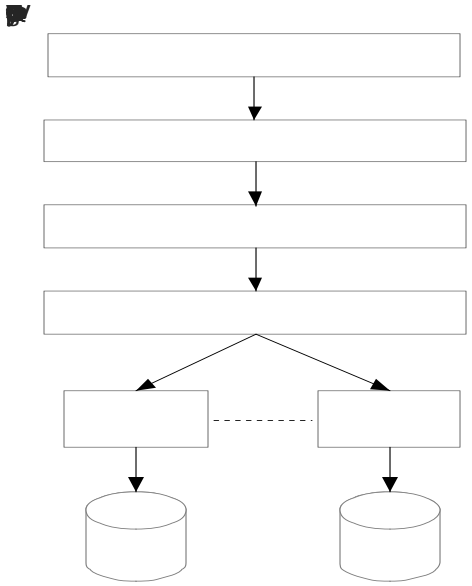

Figure 1. Data integration architecture.

Query optimization is performed as the next process in a data integration system. Though the user presents the query to the system in the defined format, the query might not always be efficient. In other words, the style of querying depends totally on the expertise of the user. The user might add unnecessary complexities to the query which might tend to reduce the efficiency of the query to a large extent. Hence it is mandatory for the system developer to incorporate an optimization engine that can perform effective translations of the query provided by the user. Difference between the traditional optimization engine and the optimization engine used in the data integration system comes from the fact that the data sources being used in the later are heterogeneous, hence the query being optimized should exploit the advantages specific to the data source that it is meant to operate on. Depending on the heterogeneity of the data sources, the complexity of optimization engine varies. Delays also happen to be the most common and unexpected in such scenarios, which is mostly constant in case of DBMS systems.

The Query Execution Engine builds the execution plan for the query. This component is rather considered

as an extension of the query optimization engine rather than a separate component in a traditional DBMS setup. But due to the heterogeneity in the data sources, this component contains different functionalities when compared to the traditional setup.

In general context, all these phases can be considered as a single block that inputs the query and returns the processed query that is eligible for operating on the specific data sources being used. Wrappers form the next layer of the data integration system that is absent in its traditional counterpart. Wrappers are the direct intermediaries to the source data. The preprocessed query, though efficient, corresponds to the format available in the data integration engine rather than the actual data source. Wrappers perform the job of converting the source data to the form that can be used by the query processor[7]. They are source specific, hence the number of wrappers to be used in an integration system depends directly on the number of data sources being used.

3. Data Integration: Challenges

This section presents the major challenges encountered while designing a data integration system. Most of these challenges have been addressed and the corresponding suggestions are presented in sections 9 and 10.

3.1 Inconsistencies due to Heterogeneity in Data Sources

The most important challenge facing a data integration system is to incorporate the heterogeneity associated with the data. Since the data to be used is from various sources, their formats vary extensively and the ability to recognize similar data or similar schemas or data itself becomes a challenging task. Further, this also leads to inconsistencies during the process of fusion, as data in one source can have a different data model when compared to other sources.

3.2 Conflicts

Conflicts refer to inconsistency in information when dealing with multiple data sources. Conflicts can take various forms such as semantic conflict, data representation conflict and data conflict[8]. Semantic based conflicts[9,10] refer to the process of identifying inconsistencies in the meanings corresponding to the data. Data representation conflicts refer to discrepancies in the data models representing similar data, and data conflicts refers to the differences within the data itself[11]. Conflicts can also arise due to inconsistent naming conventions and inconsistent data modeling used for creating the data sources. Due to the human involvement in all these operations, facing conflicts becomes an unavoidable scenario.

Mechanisms used for maintaining consistency in a single RDBMS will not be applicable in the data integration architecture. There is also no hard and fast rule providing the schematics of the conflict resolution mechanisms when it comes to the data integration framework, since the heterogeneity associated with it cannot be accurately defined. The number or type of the data sources cannot be brought into the architecture; hence conflict resolution is to be hardcoded depending upon specific applications.

Though conflict resolution appears to be a loosely bound framework, this is one of the major functionalities to be carried out in a data integration framework. The accuracy of any data integration framework depends largely on the accuracy of the conflict resolution mechanism in place.

Depending on the nature of the problem and the type of the data sources involved, the conflict resolution mechanism is identified. Hence the basic necessity for a data integration system is to be in a static state from the perspective of heterogeneity in order to resolve conflicts effectively.

3.3 Delays in Obtaining Data and Giving Results

Another major challenge facing the data integration system is the implicit and unavoidable delay associated with retrieving data from the data sources. Due to the heterogeneity, delay due to data retrieval from various data sources differs considerably. Hence the data source with highest latency acts as a bottleneck for the entire system.

4. Integrating Data: The Methods

In this study, data integration is divided into three categories, depending on the level when the integration is performed and depending on the data source where the query is applied.

4.1 Integrate Data Sources and Query

Integration of data sources can be performed initially and then querying on the integrated data source can be

performed shown in Figure 2. This method tends to provide a common interface for the queries by removing the heterogeneity associated with the data sources. Querying could be performed using a specialized query language or using a single accepted standard query language.

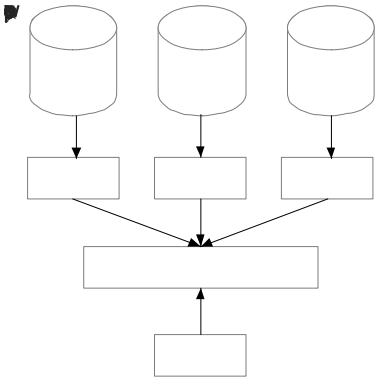

Figure 2. Integrating data sources and then query.

The process of integrating data is performed initially, which is a onetime process. Hence this mechanism will result in faster processing of the remaining queries. It tends to provide faster and more accurate results. Sizes of the data stores play a major role in determining the time taken for the integration step.

This method of processing is mostly not recommended for processing involving huge data sources. This mechanism does not blend well with dynamic data sources. In case of necessity, specialized mechanisms to detect the new data and performing necessary integration processing for combining it with the integrated data is mandatory. The process of updating information in the integrated data source is more complex when compared to the process of integration.

The major advantage of this approach is that it works best on data sources that are moderate and static but handle large amount of queries. Major structures that can benefit from this method are the static web sites with major hits. This process can also work with dynamic data sources, but it is a tradeoff that has to be balanced between frequent updating of the integrated data source and number of queries to be handled.

4.2 Depending on Query Obtain Data and Merge and then Perform Query

This process involves obtaining the data from the respective discrete sources depending on the query and then performing the integrating process with the filtered data shown in Figure 3. This method can be performed using partial data from the data source, or by retrieving the complete data from the corresponding source[12]. Obtaining partial data from the data source is a more complex process, since it involves identifying the required elements from the query and fetching it from the data sources. The process of identification of the required information is a trivial problem, hence this method requires specialized filtering methodologies. But it proves to be very useful if the data sources contained are large. The second method of retrieving the complete data source can be performed if the system operates on a large number of small data sources. This method works well with dynamic data and does not show any performance lag when compared with the static data source.

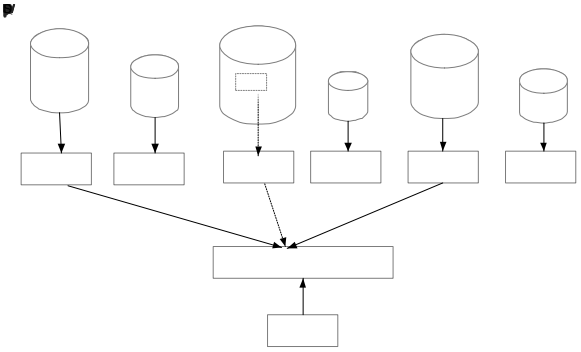

Figure 3. Query and merge data sources.

One downside of this approach is that it involves many intermediate steps for performing filtering and then integrating, which acts as additional overheads apart from the conflict resolution techniques. Further, the filtered results cannot be reused, unless a similar query is passed to the system again. Hence it is mostly discarded.

This approach can be actually improved by accumulating the results for each query to build an integrated data source, which can act as a cached system containing results for queries that have been previously processed. Similar queries can get hits if they are similar to the already posed queries, which can save considerable processing cycles and time.

4.3 Query Different Data Sources and Merge the Results

If there are too many data sources to be considered and if the data sources are large, then it becomes feasible only

to query the appropriate sources and merge the results shown in Figure 4. This is the mostly used methodology for performing data integration[13]. The efficiency of query retrieval lies entirely on the processing efficiency of the wrappers. Size of the data source and the number of data sources is oblivious to the integration engine. Due to this property, this method can operate on dynamic data sources effectively. The query is processed and the corresponding data sources are queried.

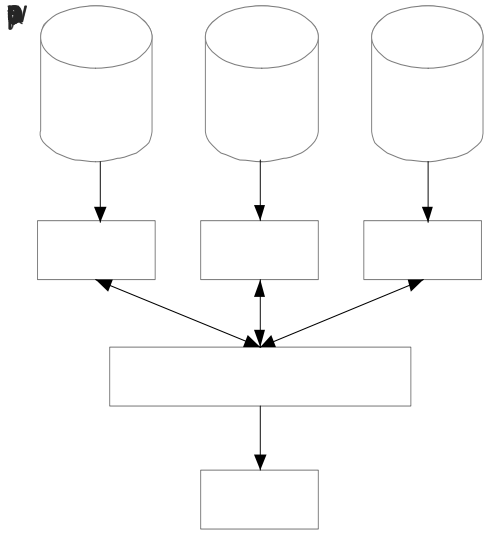

Figure 4. Query discrete data sources and merge results.

Queries are then merged and processed and the results are obtained. View based architecture is followed in this method. Hence an intermediary algorithm to convert the specialized query to the query corresponding to the native data source format is required. The data integration system can be logically projected as a database, and the results can be visualized as views corresponding to the query being posed. Since only a part of each of the related data sources is considered, this mapping is justifiable. The following algorithms describe mechanisms to perform effective projections of the results.

4.3.1 Bucket Algorithm

The Bucket algorithm[14] reformulates a user query posed on a mediated or specialized into the native format acceptable by the data source to obtain appropriate results. Here, both query and the sources are described by select-project-join queries. Every query is divided into its smallest possible sub query. Each sub query is executed and the results (views) obtained are placed in buckets. Hence for every query there exists many buckets, each containing views corresponding to its sub query. These views are finally unified and the final bucket that is formed contains the head of the view V. The algorithm then considers query rewritings, called conjunctive queries. A union of these conjunctive rewritings is finally performed to provide the results.

The bucket algorithm works by considering the context in which the sub query is presented in the system. It uses a stringent pruning mechanism that returns only the relevant results. Reusing of the query is not an option; hence a separate query is created for every process. It provides opportunities for interleaving optimization and execution in a data integration system.

5. Inverse-Rules Algorithm

The inverse-rules algorithm[14] was also formulated in the context of data integration. The major strategy followed in this algorithm is to invert the view definitions. The rules provided here aids in the computation of tuples of databases from the tuples of views.

The major advantage of the inverse rules algorithm is that it is reusable; hence the rules can be computed ahead of time. Once the inverse rules are computed, it can be applied to any query. This method does not handle arithmetic comparison predicates. The results returned by this algorithm is more general, hence should be pruned before usage. The results have high probability of containing irrelevant data. The results from the inverse rules algorithm should pass through the constraint propagation phase prior to usage.

Though both the algorithm looks similar from the aspect of computing the final results by using the divide and conquer paradigm. They have their own pros and cons which should be evaluated before shortlisting on one of them.

6. Query Reformulation Techniques

Efficiency in a data integration system can be directly mapped between the descriptions of relationships between the source relations and the mediated schema. The query process should be able to reformulate the query

presented in the intermediate schema to the query on the source schema. The following are the reformulation approaches associated with a data integration system[10].

6.1 Global As View (GAV)

The GAV approach involves the definition of the schema in a global view over the local schemas. Several local views are combined to provide a global view. A query is written for every relation R in the mediated schema specifying how to obtain R's tuples from the source.

Eg:

DB_1(id,title,actor,year)=>MOVIEACTOR(title,actor) (1)

DB_2(it,title,actor,year)=>MOVIEACTOR(title,actor) (2)

If a third data source arrives, containing reviews, then MOVIEREVIEW relation can be represented as:

DB_1(id,title,actor,year) □DB3(id,review)=>MOVIEREVIEW(title,review) (3)

The GAV descriptions are Horn Rules with relationship between the mediated schema and the source. Query reformulation is direct and simple when dealing with GAV. This is due to the fact that relations in the mediated schema correspond directly in terms of source relations. It reduces rule unfolding, hence it is easier when it comes to query reformulation. The downside of GAV is that new data source insertion is very complex. In order to add a data source, it becomes mandatory to check for all the possible relations it can have with the existing data sources. It lacks in terms of scalability, hence it is suitable for systems operating on constant data sources. The global schema needs to be rebuilt every time a source is added. TSIMMIS[15] and GARLIC[16] follows the GAV approach.

6.2 Local As View (LAV)

The LAV approach involves defining local schemas over the mediated schemas. In this method contents of the data sources are described as a query over the relations in the mediated schema. Each data source is described in isolation and their relationships are considered only in the next level.

Eg:

S1:V1 (title,year,director)⟹MOVIE (title,year,director,genre)⋀AMERICAN(director)⋀year≥1960⋀genre=comedy (4)

S2:V2 (title,review) ⟹MOVIE(title,year,director,genre)⋀year≥1990⋀MOVIEREVIEW(title,review) (5)

In LAV, Query reformulations are more complex due to the absence of direct mapping of the mediated schema over the source relations. The integration system operates to identify how the data can be combined and how they should interact with other sources. Due to this property, query reformulation becomes harder, and sometimes it requires performing recursive queries over the sources. The user has flexibility to specify rich constraints on the sources that is almost impossible in the GAV approach. This becomes mandatory when dealing with closely related and overlapping data. Hence the LAV approach is more preferred when dealing with complex and highly cohesive data sources. There is no need to remodify the global schema for every updation of the data sources. AGORA[17] follows the LAV approach. Some approaches such as PIAZZA[18] follows a combination of both GAV and LAV.

7. Ontology based Information Integration

Conflict resolution forms one of the major functionalities of any data integration system, since such a system is prone to conflicts due to the usage of a variety of heterogeneous data sources. Heterogeneity in this context is classified as structural and semantic heterogeneity. While structural heterogeneity can be sorted out by using the meta data, semantic heterogeneity poses a more serious challenge, since it deals with the meaning of the data being used. Under this context, conflicts are classified as confounding, scaling and naming conflicts.

Ontology used in a conflict resolution system is used for explicit description of the information source semantics. Three directions can be identified for employing ontology, namely single, multiple and hybrid ontology approaches[19].

The single ontology approach employs a single global ontology for specifying the semantics[20]. An independent model is built for each of the data source and they are connected to obtain the global ontology. In the multiple ontology approach, each data source uses its own specialized ontology. The hybrid approach uses a single global ontology, but it contains the description of the semantics of each source.

In addition to this, ontology also provides the query model that can function as the global query schema. It can also be used as a verification mechanism that can

automatically check the correctness of the mappings from the global schema to the local schema. Some major examples of ontology based integration methods include TSIMMIS[15], Info sleuth, KRAFT and SIMS.

8. Data Integration Languages: A Review

Traditional database queries require data transfer between a database and work area, while in a data integration system data transfer occurs between multiple databases, each having their own processing formats. Accessing the multi databases usually require a common language decipherable by the involved databases. Due to the nativity and the high usage involved with SQL, the languages designed for multi databases also have major similarities with SQL for ease of use. The following are some of the data integration languages being used.

8.1 MSQL, MDSL and SchemaSQL: A Comparison

MSQL[21], also called Multidatabase SQL, is a multi database language that expresses queries over multiple databases in a single statement. It is an extension of SQL that is formulated by adding new functions for non-procedural manipulation of data in heterogeneous and related databases. The general form of a MSQL statement is

<USE statement>

<SELECT statement>

Any function in SQL is a function in MSQL. New functions in MSQL are designed for interoperating with non-integrated SQL databases. The overall design of MSQL is based on MDSL with similarities to SQL. STORE, REPLACE and COPY statements replace the INSERT AND UPDATE statements of SQL.

MDSL[22,23] is a manipulation language of the MRDSM system. It extends the classical DML of the MRDS. Though MDSL is specific to the MRDS system, its syntax resembles QUEL and SQL. The general form of an MDSL query is

OPEN name1 [mode 1] name2 [mode2]…

RANGE (tuple_variable relation)…..

SELECT<target list>

WHERE <predicates>

The above mentioned languages use a single statement for creation or alteration in the concerned databases. Retrieval and modification are performed using join operations from the different database schemas. Broadcasting of the retrieval or modification operations to identify schematically related content helps in interoperability. They can be used to create inter-database queries for communication between databases. They help in creation of multi-database views and helps in dynamic aggregation of data. They also support triggers, stored procedures and transactions. Stored procedures usually consist of queries, hence they are called stored queries.

Query is framed by providing database name as a part of the OPEN command in MDSL, while MSQL uses the USE statement. Relation name is expressed as a part of the RANGE statement in MDSL. MSQL and SchemaSQL[24,25] represents it as a part of the FROM clause. In case of conflicts in the relation name, then it is preceded by the database name. The attributes are mentioned as a part of the SELECT and the WHERE clauses. The previous solution of appending the relation name and the database name is used in case of conflicts.

In all the above mentioned languages, the user must be aware of the database names, relation names and the attribute names in order to construct the query. Since each of these databases are designed independently, naming conflicts might occur, which will lead to problems during integration. The query languages contain built in semantics that can be used to resolve them. But this is expected to be performed manually during query construction. The semantic variables can be added as a part of the query in MDSL and MSQL. MDSL uses the RANGE_S statement, while MSQL uses LET<>BE<> statement to specify the semantic variables. In Schema SQL, in addition to the values, the context information is associated to the relation name and the attribute names.

Query processing is then performed by passing the queries represented in the intermediate languages to their corresponding data stores and converting them to their native formats for execution (as describes in Section 4).

9. Research Directions

The comprised components of a data integration system is large, hence the research directions in this area are vast and varied. Some effective research directions have been identified by us and are discussed below.

The wrapper being used for query/data conversions can be enhanced by eliminating the multiple wrappers (each for a data source) and providing an integrated wrapper for all the data sources used for integration.

Since most of the databases follow SQL based syntaxes, this will eliminate the need to write recursive mappings for most of the query structures.

In case of dynamic data stores, a wrapper generator can be created to dynamically generate wrapper clauses depending on the data store being used. This will create a generic system that can be utilized in data integration systems involving any kind of data stores. This generator can be created as a language processing engine which can be directly integrated with the query processing system of the data store for further efficiency. Classification rules can be used to identify language dependencies and the system could be programmed to function accordingly.

Optimization of the Query Optimization engine would provide effective and faster results. A mechanism that can reduce latency is to directly convert the queries directly to the required data source format and to let the native optimizer perform the optimization process. This will reduce the intermediate processing involved for optimizing the results.

When dealing with the queries as described in section IV, the process can involve detecting duplicates and storing the appropriate content by eliminating the duplicates. Semantically correlated contents can be analyzed and can be appropriately stored. This process involves dealing with conflicts in the initial stages itself. But once this process is performed, results can be provided to the users in more accurate manner with less processing latencies. Since the initial process does not involve the user's query time, it can enhance the system's performance to a large extent.

Conflict identification[26,27] is another area with a huge scope for enhancement. The process of identifying conflicting attributes when a query is presented is the basis for working with multiple databases[28]. Though this process seems simple, it has a semantic dimension to it, which increases its complexity manifold. In order to identify such attributes, machine learning methods[29,30] can be used. Further, user input can also be incorporated using standard mechanisms such as AHP to identify the best or the most important data from the set of conflicting attributes retrieved by the query.

10. Conclusion

This paper presents a detailed study of the data integration system by describing the various components involved in the system. The available query languages have been discussed in detail and their differences when compared to each other have been highlighted, which portrays the major functionalities of an intermediate query language designed for a data integration system. The research directions which act as additional add-on or improvements to the existing system have been discussed in detail. Our future research contributions will be based on these directions mentioned above.

11. References

1. Nachouki G, Quafafou M. Multi-data source fusion. Special Issue on Web Information Fusion. 2008; 9(4):523–37.
2. Ivan R, Dodero JM, Stoitsis J. Non-functional aspects of information integration and research for the web science. Procedia Computer Science. 2011; 4(1):1631-9.
3. Huimin Z, Ram S. Combining schema and instance information for integrating heterogeneous data sources. Data and Knowledge Engineering. 2007; 61(2):281-303.
4. Tao YJ, Raghavan VV, Zu Z. Web information fusion: A review of the state of the art. Information Fusion. 2008; 9(4):446-9.
5. Yu L, Huang W, Wang S, Lai KK. Web warehouse – A new web information fusion tool for web mining. Information Fusion. 2008; 9(4):501-11.
6. Wolfgang M, Lausen G. A uniform framework for integration of information from the web. Information Systems. 2004; 29(1):59-91.
7. Calvanese D, Giacomo GD, Lenzerini M, Nardi D, Rosati R. A principled approach to data integration and reconciliation in data warehousing. Proceedings of the International Workshop on Design and Management of Data Warehouses; 1999. p. 16-1-11.
8. Philipp A, Motro A. Data integration: Inconsistency detection and resolution based on source properties. Proceedings of FMII-01, International Workshop on Foundations of Models for Information Integration; 2001. p. 1-15.
9. Faraz F, Noessner J, Kiss E, Stuckenschmidt H. Mapping assistant: Interactive conflict-resolution for data integration. Poster at the 8th Extended Semantic Web Conference (ESWC); 2011.
10. Channah NF, Ouksel AM. A classification of semantic conflicts in heterogeneous database systems. Journal of Organizational Computing and Electronic Commerce. 1995; 5(2):167-93.
11. Xin Y, Zhang L, Zhong Q, Hui P. A novel method for data conflict resolution using multiple rules. ComSIS. 2013; 10(1):215-35.
12. Jens B. Data Fusion and Conflict Resolution in Integrated Information Systems. Hasso-Plattner-Institute for Software System Techniqu; 2010. p. 1-184.

13. Luna DX, Naumann F. Data fusion: Resolving data conflicts for integration. Proceedings of the Very Large Database Endowment; 2009. p. 1654-5.
14. Alon YL. Logic-based techniques in data integration. Logic-Based Artificial Intelligence. US: Springer; 2000. p. 575-95.
15. Molina G, Hammer HJ, Ireland K, Papakonstantinou Y, Ullman J, Widom J. Integrating and accessing heterogeneous information sources in TSIMMIS. Proceedings of the AAAI Symposium on Information Gathering; 1995. p. 1-4.
16. Carey MJ, Haas LM, Schwarz PM, Arya M, Cody WF, Fagin R, Flickner M, Luniewski AW, Niblack W, Petkovic D, Thomas J, Williams JH, Wimmers EL. Towards heterogeneous multimedia information systems: the Garlic approach. Proceedings of the 5th International Workshop on Research Issues in Data Engineering-Distributed Object Management (RIDE-DOM'95); 1995. p. 161–73.
17. Manolescu I, Florescu D, Kossmann D, Olteanu D, Xhumari F. Agora: Living with XML and relational. Proceedings of the International Conference on Very Large Databases (VLDB); 2000. p. 623–6.
18. Halvey AY, Ives ZG, Mork P, Tartarinov I. Piazza: Data management infrastructure for semantic web applications. Proceedings of the 12th International Conference on World Wide Web; 2003. p. 556–67.
19. Wache H, Vogele T, Visser U, Stuckenschmidt H, Schuster G, Neumann H, Hubner S. Ontology-based integration of information. A Survey of Existing Approaches. 2001. p. 1-10.
20. Heeseok J, Jeong H. Ontology-based Integration and refinement of evaluation-committee data from heterogeneous data sources. Indian Journal of Science and Technology. 2015; 8(23):1-7.
21. Witold L, Abdellatif A, Zeroual A, Nicolas B, Vigier P. MSQL: A multi-database language. Information Sciences. 1989; 49(1):59-101.
22. Markus T, Scholl MH. A classification of multi-database languages. IEEE Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems; Austin, Texas. 1994. p. 1-20.
23. Litwin W, Abdellatif A. An overview of the multi-database manipulation language MDSL. Proceedings of the IEEE; 1987; 75(5):621-32.
24. Lakshmanan VS, Sadri F, Subramanian SN. Schema SQL: An extension to SQL for multi-database interoperability. ACM Transactions on Database Systems (TODS); 2001 Dec; 26(4):476-519.
25. Lakshmanan L, Sadri F, Subramanian IN. Schema SQL - A language for interoperability in relational multi-database systems. Proceedings of the 22nd Conference Mumbai (Bombay), India, Very Large Data Base; 1996. p. 239-50.
26. Carol I, Kumar SBR. Conflict resolution and duplicate elimination in heterogeneous datasets using unified data retrieval techniques. Indian Journal of Science and Technology. 2015; 8(22):1-6.
27. Khazalah F, Malik Z, Rezgui A. Automated conflict resolution in collaborative data sharing systems using community feedbacks. Information Sciences. 2015; 298:407-24.
28. Weiguo F, Lu H, Madnick SE, Cheung D. Discovering and reconciling value conflicts for numerical data integration. Information Systems. 2001; 26(8):635-56.
29. Ramkumar T, Hariharan S, Selvamuthukumaran S. A survey on mining multiple data sources. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2013; 3(1):1-11.
30. Ramkumar T, Srinivasan R, Hariharan S. Synthesizing global association rules from different data sources based on desired interestingness metrics. The International Journal of Information Technology and Decision Making. 2014; 13(3):473-95.