RESEARCH ARTICLE OPEN ACCESS

# A Survey of Data Integration

RenHao Chen*
*( School of Information Science and Technology, Jinan University, Guangzhou, China)

----------**********----------

## Abstract:

With the continuous development of enterprises, it is difficult for all departments within an enterprise to avoid the phenomenon of "isolated islands of information" resulting from the distribution of data in various departments and the different ways of data management. The task of data integration is to integrate data from interdependent, autonomous, and heterogeneous data sources and provide users with a common query interface that enables users to access the data transparently. Instead of accessing the data source directly, the user submits the required query to the integrated system, which returns the consistent query result.

*Keywords* **—Data integration, Heterogeneous data sources, Consistency.**

----------**********----------

## I. INTRODUCTION

The business systems of all enterprises have a basic characteristic that the trading systems of each branch are independent (geographically and administratively) so that the headquarters can not technically analyze the business data of these systems in a timely manner. Therefore, with the continuous development of enterprises, it is difficult for all departments within an enterprise to avoid the phenomenon of "isolated islands of information" resulting from the distribution of data in various departments and the different ways of data management. Since the phenomenon of isolated islands of information makes it difficult for all departments to share information with each other, a large number of scholars emerged from the 90s of last century began to pay attention to data integration research. The task of data integration is to integrate the data of the interdependent, autonomous and heterogeneous data sources together to shield the data structure differences of all the data sources and provide users with a uniform query interface so that the users can use transparent Way to access these data [1]. Users do not have to know how to access heterogeneous data source data, only need to care about the information they need to query data.

Data integration system is mainly built on the basis of global mode and a series of data sources. Each data source is autonomically managed, and the global schema can be thought of as a layer of interface between the user and the underlying data source. Integrated data needs to be mapped between global mode and data source. There are two types of mapping: GAV(Global As View) and LAV(Local As View) [3]. GAV mapping represents a global pattern as a view based on a local pattern, whereas LAV mapping is from the opposite perspective, representing a local pattern as a view based on a global pattern, the details of which are discussed below.

The second section discusses two ways to integrate data, the third section describes the structure of the data integration system framework, the fourth section describes two GAV and LAV

mapping methods and compare the two mapping methods, the first section Five sections describe the integrity constraints in integrated systems. The sixth section summarizes the full text.

## II. THE WAY TO INTEGRATE DATA

An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it.

In the information integration environment, depending on whether the integrated view stores data, there are usually two types of integration methods: entity integration and virtualization integration [2].

### *A. Materialized integration*

For materialized integration, the main idea is to integrate the processed data on the integration side. In addition, in order to ensure the consistency of the data between the integrated data and the data source, data in the global mode needs to be maintained. The most typical representative of materialized integration is the data warehouse, which integrates data from multiple distributed, autonomous data sources into storage. Data may have to be cleaned due to the possibility of overlapping information and inconsistent information among data sources [6]. Data warehouse is mainly formed on the basis of ETL three processes, namely Extraction, Transformation and Load. The extraction process indicates that the operational database collects the specified data. The conversion process means that the data is converted into the specified format and the data is cleaned to ensure the data quality. The loading process indicates that the data satisfying the specified format is converted into the data warehouse after the conversion.

### *B. Virtualization integration*

The idea of virtualization is to provide users with a virtual global model that does not actually store data on the integration side. In this case, queries need to involve interaction with the underlying data source. When a user submits a query to the system, the system performs query rewriting according to the global mode-local mode mapping, rewrites the query submitted by the user based on the global mode into a query execution plan based on the underlying local mode, Get the data after the merger is returned as a query response to the user. Data integration systems are usually done in a virtualized, integrated manner in which data is actually stored only on a range of data sources. This article focuses on such systems.

## III. DATA INTEGRATION SYSTEM FRAMEWORK

A typical data integration system framework mainly includes a mediator and a wrapper. Each data source corresponds to a wrapper, and the middleware is connected to each data source through a wrapper, as shown in FIG. 3-1. The global pattern is a unified access interface provided by the integrated system to the user. The local pattern is actually an abstract representation of the data source data. After being encapsulated by the wrapper, the data of each data source has a consistent model.

The main role of the wrapper is to access the original source of information and provide standardized packaging for its data so that data from different data sources may be heterogeneous with a consistent form of data for the upper middleware further lay the foundation for further work. The middleware processes the user's query request, transforms the query based on the global schema into a subquery that the data source can process, and then obtains the corresponding subquery data from the data source accessed by the wrapper and the wrapper, and the wrapper

encapsulates the data into a consistent The model is then returned to the middleware.

Instead of accessing the data source directly, the user submits the required query to the integrated system, and the system returns a consistent query result, freeing the user from the puzzles below, what data sources the actual data is stored in, and what Way to access the data source to extract the required data. The tasks of an integrated system include deciding which data sources are relevant to the user's query, assigning query execution plans to those data sources, collecting the results returned from the various data sources, merging the results, and returning data satisfying the integrity constraints to the user.

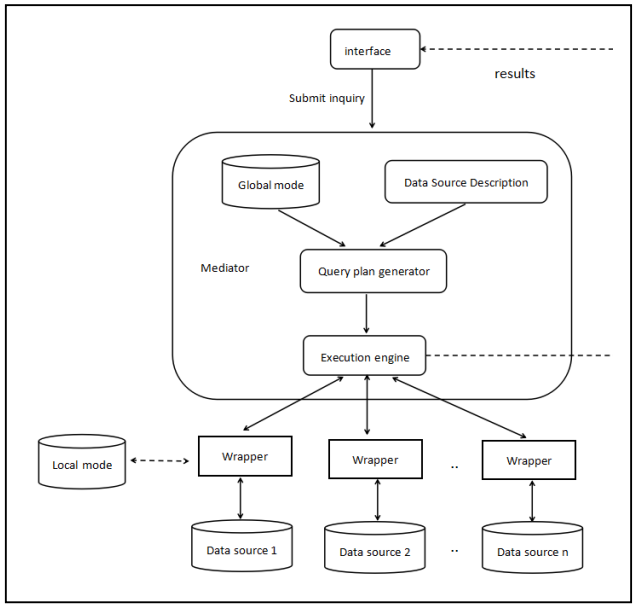


Fig. 1A framework of data integration system

The data integration system mainly works on the following three components: global mode, data source and the mapping between the two. Therefore, the data integration system I can be formalized into a triple <G, S, M>, where G Indicates the global mode, S indicates the data source, and M indicates the mapping between the global mode and the data source.

## IV. DATA SOURCE DESCRIPTION

One of the most important processes in designing a data integration system is to establish a mapping between the global schema and the data source, which determines how queries submitted to the system based on the global schema are translated into data sources. In this section we discuss two basic mapping methods: GAV(Global As View) and LAV(Local As View). We discuss these two approaches separately, and then compare the two approaches at the end of this section.

### *C. GAV*

The GAV describes a global pattern as a view based on a local pattern, which is conceptually natural because the view is usually a virtual one defined on materialized relationships. In query rewriting, it only needs to be expanded simply according to the mapping. This mapping is easy to implement. However, this mapping is not suitable for frequent changes of data sources because changing the data source can affect the mapping of other data sources.

### *D. LAV*

LAV uses the opposite view of GAV, describing the partial model as a view based on the global model. This mapping approach may seem strange, because the global mode is virtual, do not save the data, but in this mapping to adapt to the frequent changes in data sources, better flexibility. If you need to add a data source, you only need to add the data source and the global schema mapping, without the need to change other data sources.

Under LAV mapping, the local schema is described as a global schema view, and queries submitted by users based on the global schema need to be converted to queries based on partial schemas so that this can be described as a more general question, which is Overwrite the query with the view [5]. Bucket algorithm, inverse rule algorithm,

MiniCon algorithm, etc. can be used to solve this problem. Below we will briefly describe the main idea of these algorithms, the concrete realization of the algorithm can be found in related articles [7, 8, 9, 10].

***1)Bucket Algorithm***

The main idea of bucket algorithm is divided into two steps. The first step is to first create a bucket for each subquery in each query and then consider each subquery separately to determine which views may be relevant to the subquery, to be relevant to the subquery, and to be compared with the comparison predicate of the original query View into the appropriate bucket. The second step is to consider the possibility of all the combinations in the bucket, each containing one view atom in each bucket, indicating that a query is overridden and that the rewrite is included in the original query, and if so, to Rewrite focus, the final result of the bucket algorithm is the union of these rewrite.

**Example 1** Consider a query that looks for the film's more prominent movie director.

Q1(ID,Dir):-Movie(ID,Title,Year,Genre),Revenue(ID,Amount),Director(ID,Dir),Amount⩾500

Suppose we have the following view：

V1(I,G):-Movie(I,T,Y,G),Revenue(I,A),I⩾2000,A⩾600

V2(I,A):-Movie(I,T,Y,G),Revenue(I,A)

V3(I,D,Y):-Movie(I,T,Y,G),Director(I,D),I⩽1000

First, the bucket algorithm creates a bucket for each subquery in query Q1. For bucket Movie, views V1, V2, and V3 all have subqueries associated with them, so add these three views to the bucket movie. Table 1 shows the contents of each bucket:

| Movie(ID,Title,Year,Genre) | Revenue(ID,Amount) | Director(ID,Dir) |
|---|---|---|
| V1(ID,Genre) | V1(ID,G') | V3(ID,Dir,Y') |
| V2(ID,A') | V2(ID,Amount) | |
| V3(ID,D',Year) | | |

The variable with the ' in the above table indicates that there is no variable in the bucket's mapping field. The second step in the algorithm is to combine the elements in each bucket, combine the first element in each bucket, and get the following query:

q1(ID,Dir) :- V1(ID,Genre),V1(ID,G'),V3(ID,Dir,Y')

Further consider finding that the intersection of views V1 and v3 are empty sets, because they contain the IDs of disjoint movies, thus excluding this combination. Consider the following combination:

q2(ID,Dir) :- V2(ID,A'),V2(ID,Amount),V3(ID,Dir,Y')

Obviously, q2 is not included in the original query Q1 because Q1 requires Amount ⩾ 500, but we can obtain the following include override by adding the predicate Amount ⩾ 500 and deleting a redundant subquery V2 (ID, A '):

q2'(ID,Dir) :- V2(ID,Amount),V3(ID,Dir,Y'),Amount⩾500

***2) The Inverse-rules Algorithm***

The key idea of the algorithm is to construct a set of rules that reverse the definition of a view, that is, how to calculate tuples of database relations from view tuples. The following an example of the idea of the algorithm.

Consider a view of the previous example:

V(Dir,Amount) :- Director(ID,Dir),Revenue(ID,Amount)

We can construct the following inverse rules based on the view:

R1: Director(f1(Dir,Amount),Dir) :- V(Dir,Amount)

R2: Revenue(f1(Dir,Amount),Amount) :- V(Dir,Amount)

Intuitively, these two inverse rules show that if there is a tuple of (Dir, Amount) in view V, we can

deduce that there exists a constant c such that the relational data tables Director (c, Dir) and Revenue (c, Amount ), Where c is denoted by f1 (Dir, Amount) and item f1 (Dir, Amount) is called SKolem [7], indicating a certain constant depending on the values Dir, Amount and function f1.

**Example 2** Suppose there is such a query, find out the director Jack's income per film,

Q(Amount) :- Director(ID,Jack),Revenue(ID,Amount)

And we know that view V contains three tuples: {(Jack, 5000), (Jack, 8000), (Tom, 6000)}, we can get the following tuple according to the inverse rule:

Director：

{(f1(Jack,5000),Jack) , (f1(Jack,8000),Jack) , (f1(Tom,6000),Tom)}

Revenue：

{(f1(Jack,5000),5000) , (f1(Jack,8000),8000) , (f1(Tom,6000),6000)}

The query Q is applied to the above extension, the result of the query can eventually be 5000 and 8000.

***3)MiniCon Algorithom***

The beginning of the MiniCon algorithm is somewhat similar to the bucket algorithm, considering which views have subqueries related to subqueries in the query. But unlike bucket algorithms, once a partial mapping of subquery g of query Q to a partial query g1 of view V is found, the connection predicate is shifted to the query Q (ie multiple occurrences in the subquery Variable) and find the minimal additional subquery set that you need to connect with view V to rewrite query Q. The mapping information of these variables and the minimal additional sub-query set constitute a MCD (MiniCon Description). The second step in the algorithm is to combine these MCDs to generate query rewrites. Compared with the bucket algorithm, the MCD is constructed in a different way and does not require inclusion checking in the second stage, so it is more efficient than the bucket algorithm. The following example illustrates the idea of MiniCon algorithm.

**Example 3** Consider the following query, check out the movie starring movie director information.

Q3(Title,Year,Dir) :- Movie(ID,Title,Year,Genre),Director(ID,Dir),Actor(ID,Dir)

And given the following view:

V1(D,A) :- Director(I,D),Actor(I,A)

V2(T,Y,D,A) :- Movie(I,T,Y,G),Director(I,D),Actor(I,A)

The bucket algorithm creates a bucket for each subquery in query Q3, and view V1 is added to the two buckets, Director (ID, Dir) and Actor (ID, Dir). However, a careful analysis reveals that in fact view V1 is not useful for query rewriting, because view V1 is useful and must be linked to Movie (ID, Title, Year, Genre) and join predicate 'I' does not appear In the head of V1 In the above example, the MiniCon algorithm can find that view V1 can not be used for query overwriting, so MCDs are no longer created for it. Create an MCD for view V2, {A-> D, V2 (T, Y, D, D), Title-> T, Year-> Y, Dir-> D, {1,2,3}}. The second phase is to combine MCDs so that all subqueries in the query are overwritten to create conjunctions and output the union of the conjunctions.

***4)Algorithom comparison***

The advantage of the bucket algorithm is that it uses the number of elements in a bucket of predicates compared to the query, effectively reducing the amount of rewrite that needs to be considered. However, bucket algorithms do not consider the interactions between different subqueries in queries and views, so buckets may

contain views that can not be used for rewriting, reducing efficiency. MiniCon algorithm overcomes this problem, and because of the way of constructing MCD, the second phase of MiniCon algorithm does not need to include check, and further improves the efficiency. The advantage of the inverse rule algorithm is conceptually simple, it is based on the logic of the method of reverse. From the efficiency to consider, MiniCon better.

### *E. Comparison of two mapping methods*

Query rewriting under GAV mapping only needs to expand the original global pattern-based query according to the rules, which is simple and direct. However, when the system needs to increase or decrease the data source, it is very inflexible because the data source is increased or decreased This means that the global schema needs to be redefined.

Mapping a new data source to LAV implies adding a new view definition to that data source, leaving other data sources unaffected, so adding or subtracting data sources is easier and more flexible, but the associated algorithms Corresponding to more complicated.

## V. DATA CONSISTENCY IN INTEGRATED SYSTEMS

The integrity of the database refers to the correctness and rationality of the data. It reflects the original appearance of the entity in the real world. Therefore, whether or not the database has completeness determines whether it truly describes the real world or not and whether it makes any sense on its own. In order to maintain the integrity of the data, the database management system (DBMS) must provide a mechanism to check that the data meets the given constraints. In essence, a data integration system can also be thought of as a DBMS, and it must also ensure that the data satisfy the integrity constraints so that valid data is returned for queries submitted by the user. There is a case where some data, even after satisfying their respective data source integrity constraints, may violate the integrity constraints defined in the integrated global schema, resulting in inconsistent DB instances based on the global schema, After the data consistency maintenance. However, for many reasons, integrity constraints may not be enforced or satisfied. For example, views in virtualization integration that provide queries do not actually save data, so the view's integrity constraints do not actually act on the data, but the integrity constraints are handled during the query. This raises the question of how to obtain consistent query results from a database that does not satisfy consistency.

### *F. Data repairs*

One way is to repair the original inconsistent database, making the database data in a state of consistency, and then from a consistent database query. Here involves a concept of distance. Given a database instance r, we denote by $\sum(r)$ the set of formulas $\{ P(\bar{a}) \mid r \models P(\bar{a}) \}$, where P is the relation name, $\bar{a}$ is a tuple.

**Difinition 1**.The distance $\triangle$(r,r') between database instances r and r' is the symmetric difference:

$$\triangle(r,r')=(\sum(r)-\sum(r'))\cup(\sum(r')-\sum(r))$$

**Difinition 2**.For the database instance r, r', r'',

$r' \leq_r r''$ if $\triangle(r,r') \subseteq \triangle(r,r'')$, i.e., the distance between r and r' is less than or equal to r and r "distance.

**Difinition 3**.Given a database instance r and r', r' is a repair of r if $r' \models IC$ and $r' \leq_r$-minimal, i.e., r 'is a database instance that satisfies the integrity constraint and has the smallest distance from r.

**Example 4**Consider two tables, D1 and D2, that have a table Student, with the primary key ID, for each of them fulfilling its primary key constraint:

D1.Student：

| ID | Name | nationality |
|---|---|---|
| 1001 | Jack | US |
| 1004 | Tom | China |
| 1006 | Jame | UK |

D2.Student：

| ID | Name | nationality |
|---|---|---|
| 1001 | Paul | US |
| 1005 | Cathy | China |

When we integrate these data, we get a table that does not satisfy the consistency:

Student：

| ID | Name | nationality |
|---|---|---|
| 1001 | Jack | US |
| 1001 | Paul | US |
| 1004 | Tom | China |
| 1005 | Cathy | China |
| 1006 | Jame | UK |

Based on the idea of repair, two repairs for the Student table are:

Student'：

| ID | Name | nationality |
|---|---|---|
| 1001 | Jack | US |
| 1004 | Tom | China |
| 1005 | Cathy | China |
| 1006 | Jame | UK |

Student"：

| ID | Name | nationality |
|---|---|---|
| 1001 | Paul | US |
| 1004 | Tom | China |
| 1005 | Cathy | China |
| 1006 | Jame | UK |

***G. Obtain consistent data directly***

Data repair techniques attempt to identify and correct errors in the data and can be used to restore the database to a consistent state. However, the fix is preferably semi-automatic and may not be feasible or not acceptable for some applications. In addition, a single repair strategy may not be suitable for some environments, users may want to try different repair strategies, or may retain all data, including even inconsistent data. In [12], a method is proposed that can directly rewrite queries to obtain consistent data directly from an inconsistent database and apply the method to a ConQuer system. Here's a simple example to explain his ideas.

**Example 5** Consider the following does not meet the consistency of the database, store the user's ID custID and the user's account balance balance, where custID is the primary key. Note that the following table violates the primary key constraint, probably because its data comes from more than one data source.

| | custID | balance |
|---|---|---|

| | | |
|---|---|---|
| t1 | c1 | 500 |
| t2 | c1 | 3000 |
| t3 | c2 | 2600 |
| t4 | c2 | 2800 |
| t5 | c3 | 4000 |

Consider a query that requires the user to find out that the account balance is more than 2000:

Q4：select custID from CustBalance where balance>2000

If you query directly on the table, you get {c1, c2, c2, c3}. c1 appears in the above result is because c1 has another account balance to meet the query Q4, that is, t2. However, closer examination reveals that user c1 has an account balance below 2000, as shown by tuple t1, so c1 should not appear in the query result. In addition, user c2 appears twice in the result. In this example, the expected query result that satisfies the consistency should be {c2, c3}. Although there are two accounts for user c2 in the table, the balance of both accounts satisfies the query Q4 condition, so As a result of the inquiry.

For the sake of consistency, the ConQuer system rewrites the original query, and for the example above it will produce the following rewrite:

```
select distinct custID
from CustBalance as cb
where balance > 2000 and
    not exists (select *
    from CustBalance as cb'
    where cb'.custID=cb.custID and cb'.balance
    ≤2000 )
```

The rewritten query execution in the original does not meet the consistency of the data table, we can get the desired results {c2, c3}. In the rewritten query, the keyword distinct is used to exclude duplicate elements in the result, and a nested subquery uses the keyword not exists to filter out inconsistent data and eventually obtain consistent data. The realization of specific algorithm can refer to [12].

## VI. CONCLUSION

This article describes the overall architecture of the data integration system, as well as some of the key modules of the algorithm description and comparison. After years of research, data integration technology has also been more and more applied to various fields. Today, integrated data sources are also being extended to unstructured Web data by traditional structured data such as relational databases, semi-structured data such as XML. We believe that with the continuous development of computer technology, some problems in data integration will be solved very well, and the application of data integration will also be more extensive.

## REFERENCES


[1] CHEN Yue-Guo, WANG Jing-Chun, A Review of Data Integration, Computer Science,2004,vol.31.

[2] ZhangHai Li, Research on Integrity Constraints in Integrated Data,JiNan University,2016.

[3] Maurizio Lenzerini, Data Integration: A Theoretical Perspective, Symposium on Principles of Database Systems , 2002 :233-246.

[4] Diego Calvanese, Giuseppe De GiacomoData, Integration: A Logic-Based Perspective, Ai Magazine , 2005 , 26 (1) :59-70.

[5] L Bertossi, L BravoConsistent, Query Answers in Virtual Data Integration Systems, Inconsistency Tolerance , 2005 , 3300 :42-83.

[6] PA Bernstein ,LM Haas, Information integration in the enterprise,ACM , 2008 , 51 (9) :72-79.

[7] AY Halevy, Answering queries using views: A survey, Vldb Journal, 2001 , 10 (4) :270-294.

[8] R Pottinger， A Halevy, MiniCon: A scalable algorithm for answering queries using views, Vldb Journal , 2001 , 10 (2-3) :182-198.

[9] AY Halevy, Theory of Answering Queries Using Views, ACM , 2000 , 29 (4) :40-47.

[10] AY Levy, A Rajaraman, JJ Ordille, Querying Heterogeneous Information Sources Using Source Descriptions, Stanford Infolab, 1996 :251--262.

[11] Arenas M, Bertossi L, Chomicki J. Consistent query answers in inconsistent databases[C]// 1999:68-79 .

[12] Fuxman A, Fazli E. ConQuer: efficient management of inconsistent databases[C]// ACM SIGMOD International Conference on Management of Data. ACM, 2005:155-166.