

A Project Report  
On  
**Integrating Relational Databases**  
**BITS, Pilani-Hyderabad Campus**  
By

**UDDHAV MISHRA**  
**2014A7PS048H**

Under the supervision of  
**Prof. R. GURURAJ**

**SUBMITTED IN COMPLETE FULFILLMENT OF THE REQUIREMENTS OF**  
**CS F366: LAB ORIENTED PROJECT**



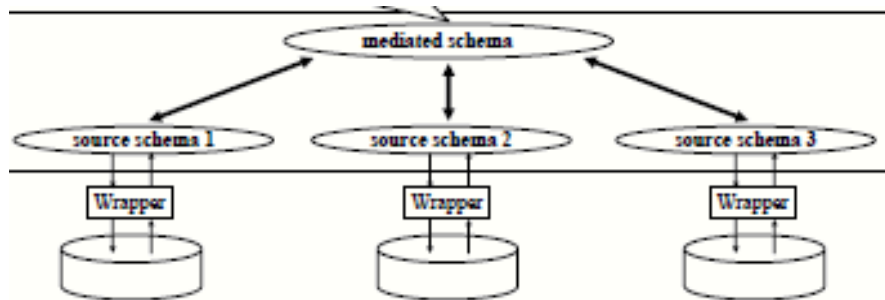
**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)**  
**HYDERABAD CAMPUS**

## **ACKNOWLEDGEMENT**

I owe my sincere gratitude to Prof. R. Gururaj, a distinguished faculty at BITS, Pilani - Hyderabad Campus and my mentor for this project, for his constant guidance, invaluable suggestions and advices at every stage of the project.

## ABSTRACT

Data Integration is necessary these days as the volume of data is rapidly increasing. So data residing in various databases has to be combined. A smart approach to be above problem is not to manually combine the databases into a single database, but providing the user an illusion that the data is combined by allowing him to query on a mediated schema and displaying the results accordingly. The mediated schema has the semantic mappings which when queried upon automatically directs the queries to the respective databases and displays the result accordingly.



# CONTENTS

Title page.....	1
Abstract.....	3
1. Source Schemas.....	5
2. SQL query on mediated schema.....	6
3. Integrating Heterogeneous databases.....	7
4. Schema of an XML document(DTD).....	8
5. Structure of an XML document.....	9
6. Mapping DTD to relational schema.....	10
7. Getting Relational Schema from DTD.....	11
8. Storing XML data into a Relational database.....	13
9. XML as mediated Schema.....	14
10. XPath query on XML document.....	15
11. Improvements and further work.....	17
12. Chosen technologies.....	18
Conclusion.....	19
References.....	20

## Source Schemas

The source schema is a schema which store the information pertaining to a particular database.

The below mentioned are two such source schemas. They store the information regarding a bookstore.

The details of the books are in two different schemas.

*Source Schema 1*

	title	author	price	year_published	category
1	introduction to algorithms	thomas comen	1200	2000	algorithms
2	artificial intelligence	rusell norvig	1000	2004	artificial intelligence
3	discrete structures	ross	400	1994	discrete strucutres
4	operating system concepts	mike	350	2008	operating system

*Source Schema 2*

	book_name	writer	cost	year
1	discovery of india	jawahar lal nehru	150	1950
2	3 mistakes of my life	chetan bhagat	250	1999
3	hary potter	j k rowling	520	1998

## SQL QUERY ON MEDIATED SCHEMA

```
alter procedure total
as
declare @n1 nchar(50);
declare @n2 nchar(50);
declare @n3 nchar(50);
declare @n4 nchar(50);
declare @n5 int ;
declare @n6 nchar(50);

declare c1 cursor static for select dab_name,table_name,price,title from ANS.dbo.tb1;
open c1;
if @@cursor_rows>0
begin
fetch next from c1 into @n1,@n2,@n3,@n4
while @@Fetch_status = 0
begin
exec('declare c2 cursor static for select ' + @n3+', '+@n4+' from '+@n1+'.dbo.'+@n2);
open c2
if @@cursor_rows>0
begin
fetch next from c2 into @n5,@n6
while @@Fetch_status = 0
begin
if @n5>500 print(@n6)
fetch next from c2 into @n5,@n6
end
end
close c2
deallocate c2
fetch next from c1 into @n1,@n2,@n3,@n4
```

```
exec total
```



Messages

```
introduction to algorithms
artificial intelligence
harry potter
```

## **Integrating Heterogeneous Databases**

Data might be residing in many types of databases like relational database, object database, XML document...etc. Consider a situation in which there is a need to combine the data in a bookstore having two different branches. One branch might be using a relational database to store the data and the other branch might be using XML document. This is a typical example of heterogeneous database integration.

The main idea in such a situation is to come up with a smart unified/mediated schema. The mediated schema is like a virtual database which does not really contain data but has wrappers which direct the query the user enters to their respective databases.

## Schema of a XML Document(DTD)

The schema of an XML document is called Document Type Definition(DTD). The DTD defines the structure of the document as shown below. It also gives information regarding the attributes .

Document Type Definition(DTD) is very easy to write and is used to represent the entire XML document. It can be included in an XML file or can be a separate document.

```
1  <!ELEMENT person (name, age, address)>
2  <!ELEMENT name (fname, lname)>
3  <!ELEMENT fname (#PCDATA)>
4  <!ELEMENT lname (#PCDATA)>
5  <!ELEMENT age (#integer)>
6  <!ELEMENT address (house, street, city)>
7  <!ELEMENT house (#PCDATA)>
8  <!ELEMENT street (#PCDATA)>
9  <!ELEMENT city (#PCDATA)>
10 <!ELEMENT alien (name,age)>
```



## Structure of a XML Document

The XML document contains data between `<>` and `</>` tags where the data inside the tags represent the attribute name.

For example, in the figure shown below:

Book	->	Table name
Booktitle	->	Column name
The Selfish Gene	->	Cell value

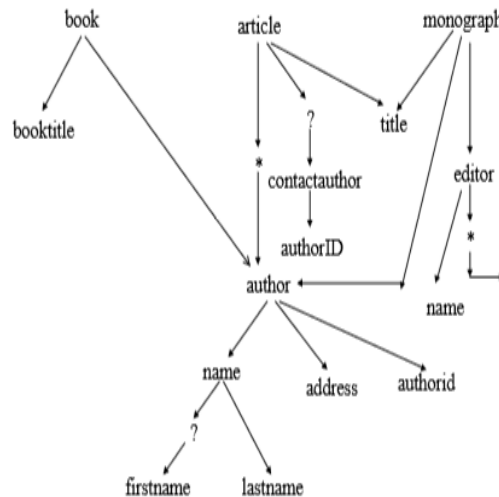
```
<book>
  <booktitle> The Selfish Gene </booktitle>
  <author id = "dawkins">
    <name>
      <firstname> Richard </firstname>
      <lastname> Dawkins </lastname>
    </name>
    <address>
      <city> Timbaktu </city>
      <zip> 99999 </zip>
    </address>
  </author>
</book>
```

## Mapping DTD to Relational Schema

DTD can be converted to relational schema by using the inlining technique. The idea used was to doing a Depth First traversal of the DTD graph and combining the descendants into a single relation.

So, initially we converted the DTD schema into a graph structure as shown below and then did a Depth First traversal of the graph to get the relational schema.

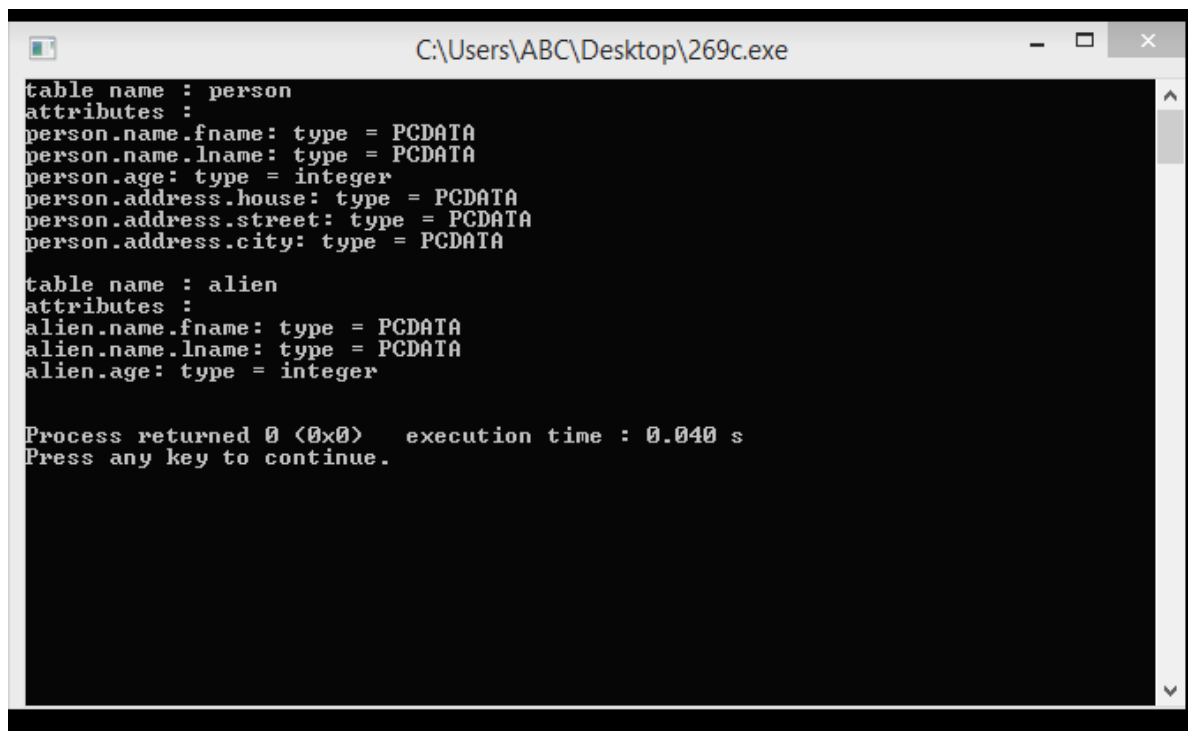
```
<!ELEMENT book (booktitle, author)
<!ELEMENT article (title, author*, contactauthor)>
<!ELEMENT contactauthor EMPTY>
<!ATTLIST contactauthor authorID IDREF IMPLIED>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor (monograph*)>
<!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT author (name, address)>
<!ATTLIST author id ID #REQUIRED>
<!ELEMENT name (firstname?, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT address ANY>
```



## Getting Relational Schema from DTD

The code takes input as a DTD and gives output all the tables that the given DTD represents. PCDATA represents string data type of variables. All the nodes are numbered from 1 to n in order of their appearance in DTD. The graph is then created using vectors. DFS function is used to traverse DTD for all the individual connected components. For nodes numbered 1 to n new DFS started whenever a node with in-degree equal to 0 is encountered. In DFS function a string is also passed that carries the value of string till that position. So the child of a node would contain values of strings of all its ancestors with separated with a dot. The function is recursively called for all children of a particular node that is under consideration at that moment. One important thing to observe is that each connected component of graph is treated as a table with table name as the root node of that component which also has in-degree of zero and the leaf nodes are treated as attributes of the specific table. Nodes other than root nodes are not counted as visited once they are visited so that if some other component tries to merge with current connected component those values can also be seen for that table.

```
1 <!ELEMENT person (name, age, address)>
2 <!ELEMENT name (fname, lname)>
3 <!ELEMENT fname (#PCDATA)>
4 <!ELEMENT lname (#PCDATA)>
5 <!ELEMENT age (#integer)>
6 <!ELEMENT address (house, street, city)>
7 <!ELEMENT house (#PCDATA)>
8 <!ELEMENT street (#PCDATA)>
9 <!ELEMENT city (#PCDATA)>
10 <!ELEMENT alien (name,age)>
```



```
C:\Users\ABC\Desktop\269c.exe

table name : person
attributes :
person.name.fname: type = PCDATA
person.name.lname: type = PCDATA
person.age: type = integer
person.address.house: type = PCDATA
person.address.street: type = PCDATA
person.address.city: type = PCDATA

table name : alien
attributes :
alien.name.fname: type = PCDATA
alien.name.lname: type = PCDATA
alien.age: type = integer

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

## Storing XML data in a Relational Database

After we got the relational schema, using the inlining technique, now we processed the XML document which contained the data, and inserted the data into a relational database. This was done by importing `pysql` package in python and connecting to a localhost and creating a connection. After the connection is created, we inserted the tuples into the database.

```
conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='', db='test', charset='utf8mb4', autocommit=True)
```

## XML as the Mediated Schema

In the heterogeneous integration problem there must be a mediated schema on which the user queries, to get his results. The mediated schema is like a combined database which has wrappers to the respective database. The most commonly used mediated schema must be in XML format as XML supports nesting of data. But instead if we had chosen relational schema as the intermediate schema the major drawback would be that since relational database does not support nesting, so the user could not get answers for some of his queries.

So, for the problem of integrating an XML database and a relational database a unified XML schema would be created. This schema is hardcoded as of now, but it could be automatised using Machine learning techniques. So, the user thinks that there is only one database i.e the mediated schema and queries on it. The mediated schema in turn smartly splits the query and sends it to the respective databases. So, in this case if the user gives a XPath query, the mediated schema redirects the same query to the XML database and for the relational database it converts the XPath query into SQL. It gets the result from both the databases, combines them and shows the result to the user.

```
<!ELEMENT book(title,price,author)>
<!ELEMENT title(#PCDATA)>
<!ELEMENT price(#integer)>
<!ELEMENT author(#PCDATA)>
```

Sample Mediated Schema

## Xpath query on an XML document

Xpath is used to query the XML documents. It is similar to what SQL is for relational databases.

Example query is shown the figure below.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title>bookname</title>
    <price>cost</price>
    <author>writtenby</author>
    <filename>book_details</filename>
    <initialinf>book</initialinf>
    <dbname>null</dbname>
    <tname>null</tname>
  </book>
  <book >
    <title>book_title</title>
    <price>rate</price>
    <author>writer</author>
    <filename>null</filename>
    <initialinf>data/book</initialinf>
    <dbname>book_manager</dbname>
    <tname>books</tname>
  </book>
</bookstore>
```

The above figure corresponds to the mediated schema on which the user gives his query.

```
==== RESTART: C:\Users\ABC\Downloads\c.py =====
input xpath query relative to dtd provided
sample xpath query is given for reference:
book/price[text()<=34]/parent::book/title
book/price[text()>=34]/parent::book/title
os_concepts
helloworld
('discovery of india',)
>>> |
```

The above figure corresponds to the XPath query given by the user on the mediated schema.

In this technique of heterogeneous data integration we can achieve integration for xml and relational databases by constructing a mediated schema that is in xml format. User will only be able to see the DTD of mediated schema and will query in xpath format on that xml-document. The python code reads the query supplied by user in xpath format and gets all entries from mediated schema and get results from each individual database .Each object instance in mediated schema represents a database and corresponding values with that. The python code serves as an intermediate wrapper that decomposes the original query and supplies specific query to each data base accordingly. The wrapper created by us works for xpath queries that are in specified format. after reading the query the code iterated through all databases. Apart from the attributes of the original data this schema would contain additional attributes such as database name, table name, initial information and filename.dbname and tname would be null for xml data and initialising and filename null for relational database. This way databases are easily distinguishable. If the database is xml then the given original xpath query is modifies using the values in mediated schema and is then supplied to the original physical database file.xpath(Str) gives the output when str is executed as xpath query on xml file. Similarly in case of relational database query is broken down into 3 parts select part, from part and condition part. Using string manipulation each part is extracted from the original x-path query and then mysql query is supplied to database:dbname and table:tname . Results achieved individually are printed to get overall result. This technique gives user an illusion of presence of integrated database even though no real physical integrated database is present. The original query and mediated schema is used to get individual queries from the original supplied query to achieve data integration.



## **Improvements and Further Work**

Instead of hardcoding the mediated schema, we could use machine learning techniques to get this schema automatically.

Also if the query given by the user is more complicated the corresponding code has to be modified a little taking into consideration all possible cases.

## **Chosen Technologies**

The main technologies used in this project were:

- SQL
- Microsoft Workbench
- Python
- C++
- XML
- XPath

## **Conclusion**

In this way both the homogeneous and heterogeneous integration problem could be solved without directly combining the databases using the concept of virtual mediated schema. The major benefit from of this approach is that eventhough any new database comes up or any existing database has to be removed, we can simply remove it's corresponding entry in the mediated schema.

## References

- <http://pages.cs.wisc.edu/~whshen/papers/mobs-icde05-poster.pdf>
- <http://marenas.sitios.ing.uc.cl/publications/hyperion.pdf>
- [https://en.wikipedia.org/wiki/Data\\_integration](https://en.wikipedia.org/wiki/Data_integration)
- Integrating XML and Relational Database Systems by Gerti Kappel.
- Relational Databases for Querying XML Documents: Limitations and Opportunities Jayavel Shanmugasundaram Kristin Tufte Gang He Chun Zhang David DeWitt Jeffrey Naughton.