4.0 Introduction

The query languages discussed in Chapter-1 expect the user to have knowledge about the structure of data, which he/she wishes to access, present in one or more component databases. The user uses this knowledge to frame queries in order to access data present in multiple component databases. The user while framing the query explicitly specifies the names of the databases, the relations and the attributes therein that he/she wants to access. In case of conflicts in relation names, the relation name is prefixed by the corresponding database name whereas in case of conflicts in the attribute names, the attribute name is prefixed by the corresponding relation name which in turn may be prefixed by the corresponding component database name. This knowledge regarding the databases, the relations in these databases, and the attributes in these relations, requires that a user have extensive knowledge about the global system. Further, in a multidatabase system as discussed in Chapter-1, the component databases are autonomous and hence can associate and disassociate from the global system at any time, making this task even more difficult. This problem
exists even while accessing single database systems but gets compounded when dealing with multiple component database systems. Hence, there is a need to lessen the user burden while specifying the query.

This problem can be handled by providing a simplified query interface to the user for framing the query. The user should be able to pose a query without needing to know about the component databases registered with the global system. The query interface should assist the user by providing some minimal information about these component databases to frame the query.

A structure independent query language (SIQL) is proposed [Parimala2002a] [Parimala2002b] to provide a simplified query interface to access multidatabase systems. The query language has no explicit reference to the structure of the database i.e. the query is specified independent of all path expressions. It is up to the global system to identify the structures from which the data is to be fetched. These structures which are absent in the query are inferred at run-time by using the multidatabase dictionary present in the global system and the mapping information of the component databases registered with the global system. The structures to be identified include the schema names and the relation names that are required to answer the user query. These structures so identified are used by the global system to arrive at the generated query for the user query. The generated query is subsequently executed whereafter the result is presented to the user.

The overall approach used to arrive at the result based on the user query is discussed next.
4.1 Approach

The user, in our system, is not expected to be familiar with the structure of the component databases while framing the user query. Even though the user query is independent of all the references to the structure of the databases, it must be expressed using the database names. These database names are the SMNs that represent the relations and attributes present in the component databases registered with the global system. In order to facilitate the user in framing the query with SMNs, we provide the user with a graphical user interface (GUI). The GUI lists the SMNs and the user clicks on the ones, which are to form part of the query. The user query is expressed in SIQL. A name in the user query may be defined in more than one component database i.e. a name in the user query can be mapped to different structures in the various component databases each giving rise to a different query. We generate queries, called the generated queries, corresponding to each of these definitions. The generated queries are ordered using the 'closeness' criterion wherein the query, which is the most preferred one, from the user's point of view, is highest in the order. The 'closeness' criterion is expressed using two principles based on the Distance and the Weighted Count discussed later. The query which has minimum Distance and maximum Weighted Count is highest in the order of preference. We propose to generate queries in the order defined by 'closeness'.

The architecture of the Query Generator, which generates generated queries according to the desired order, is shown in Figure-1.
Figure-1 – Query Generator Architecture
The user query in SIQL consists of the SMNs used by the global system. These SMNs need to be resolved in order to find the structures in the component databases that can be used to answer the user query. The Reference Table Creator, using the Multidatabase Dictionary in the global system and the Mapping Information of the component databases registered with the global system, creates a temporary table, called the Reference table. The Reference table stores information about the references to structures in different component databases, for the names, in the user query. This information is in terms of the schema names and the relation names. The Meta Query Generator generates Meta queries for this table. The Resolution Finder executes the meta query. The result of executing a meta query gives the structure information, referred to as a Query Resolution, for the names in the user query. The structure information is in terms of the schema name and the relation name. The Generated Query Generator uses the mapping information present at the component databases referred to in the resolution to transform the Query Resolution into a Generated Query. This Generated Query is executed by the Query Processor to give a result to the user query.

4.2 The Query Language

The query language SIQL so proposed enables a user to pose a query with no explicit reference to the structures in the component databases. Unlike the query languages discussed in Chapter-1 the user is not expected to know the structure of the component databases while posing a query. The query language interface of SIQL is similar to SQL[Date1997]. The query is expressed using the SELECT
clause but with the FROM clause deleted. In other words, the form of the query in SIQL is as follows:

\[
\text{SELECT } \text{name}_1, \text{name}_2 \ldots \text{name}_n \\
\text{WHERE } \text{condition}
\]

where \( \text{name}_1, \text{name}_2 \ldots \text{name}_n \) are the semantically meaningful names present in the global system. Each of these names may represent a relation and/or an attribute present in the component databases registered with the global system. These names are assigned by the global system SIQS. Since these names may represent a relation or an attribute, or both, the relation name and the database name do not prefix them.

The component databases to be accessed are also not specified in the user query.

The condition is a Boolean expression which is specified using AND, OR and NOT. The names in the condition are also free from qualification.

A Graphical User Interface is built for providing the user with all the SMN that represent the relations and attributes, which are present in the various component databases that are registered with the system. This interface helps the user in forming the query. The graphical user interface to frame a query in SIQL is discussed next.

### 4.2.1 Graphical User Interface

In order to help the user construct a query, the user is provided with a graphical interface. The GUI is provided to the user in the form of a window in which the query can be constructed. The window is shown in Figure-2.
The window has two slots - one for specifying the names as part of the Select (SELECT clause) and the other for specifying the Condition (WHERE clause). Since the user initially is unfamiliar with database names, the relation names and the attribute names, the list of SMNs, representing the relations or attributes or both is provided to the user. The window presents two lists, one each for the SELECT clause and the WHERE clause. These lists of SMNs are provided as part of the list box presented in the window. The list of names for the SELECT clause consist of all the SMNs present in the global system whereas the list of names for the WHERE clause consist of the SMNs that represent attributes that are present in all the component databases registered with the global system. The user clicks on a name in order to select it. Once the query is constructed the user clicks on the Submit button. The user is also given a choice to quit the global system by clicking the Quit button.
4.2.2 Generating Queries

The user frames an SIQL query and submits it to the SIQS for execution. Since the names in the user query have no explicit reference to the structures in the component databases registered with the global system, the structures need to be identified by SIQS. The structures are in terms of the schema name and the relation name. The multidatabase dictionary in the global system and the mapping information of the component databases registered with the global system are used to identify these structures. The names in the user query are compared with the SMNs in LRMI, discussed in Chapter-3, of the component databases, to find the corresponding local relation names. These local relation names along with schema names form the structures.

In general there can be one or more set of structures that can be used to answer the user query. That is, more than one query with path expressions can be generated for a given user query. Each of these sets of structures is referred to as a resolution.

Let the user query be

\[
\text{SELECT Ename, Dname}
\]

Using the schemas shown in Figure-14 of Chapter-3, the above query can be answered using the following two resolutions

a) the Employee and Department relations of Schema_1, and

b) the Employee relation of Schema_1 and Dept relation of Schema_2.
The corresponding generated queries are

(a) SELECT Schema_1.Employee.Ename, Schema_1.Department.Name
    FROM Schema_1.Employee, Schema_1.Department
    WHERE Schema_1.Employee.D# = Schema_1.Department.D#

(b) SELECT Schema_1.Employee.Ename, Schema_2.Dept.Dname
    FROM Schema_1.Employee, Schema_2.Dept
    WHERE Schema_1.Employee.D# = to_char(Schema_2.Dept.Dno)

Since both queries need to be generated simultaneously which of them is generated first is an issue that needs to be resolved. The basis of generating one query before the other should correspond to the most desired one from the user's point of view. If we look at the example above, we find that the user might prefer resolution (a) to (b). This is because resolution (a) and the corresponding generated query involve a lesser number of schemas. This preference is expressed using the notion of 'closeness'. If the generated query involves only one relation, then it is very 'close'. At the other extreme is the query, which spans maximum schemas and maximum relations. That is, the generated queries which involve lesser number of relations (and therefore, lesser joins) and lesser schemas are more 'close' than those which involve more joins and schemas. This 'closeness' criterion can be used to order the generated queries with the ones more 'close' higher in the order. This 'closeness' criterion is defined using two principles - density and relationship. The two principles are discussed next.
4.2.3 The Two Principles

When the user specifies a query, as shown above, there can be more than one resolution for a given user query. We believe that the users prefer certain resolutions to others. If all the names are resolved to a single relation of a schema, then that resolution is the most preferred one. If that is not possible, then the next 'close' resolution is the one where all of them belong to one schema. Within these, the query resolution with lesser number of joins is more 'close' than the one with more joins. If none of these is possible, then data from multiple schema is to be retrieved. This desirability is expressed using two principles - density and relationship. These principles are explained below.

The 'density' principle states that if a field has multiple definitions then, the resolution of a name to the structure that has been referred to maximum number of times in this query resolution is more close than others which have been referred to lesser number of times. As an example, consider the query

```
SELECT Company, Cname, Dname, Cloc
```

Let us assume that Company has been resolved to Schema_3.Industry and Cname and Dname have been resolved to Schema_3.Branch. Cloc can be resolved to either Schema_3.Industry or Schema_3.Branch. Schema_3.Industry has been referred to once in this query resolution whereas Schema_3.Branch has been referred to twice. Therefore, resolution of Cloc to the latter is more close than its resolution to the former.
The 'relationship' principle states that if a field has multiple definitions then the resolution of a name to a structure in the query which gives rise to minimum number of joins is more close than that structure where the number of joins are more. As an example, consider the query

```
SELECT E#, Cname
```

Let us assume that E# has been resolved to in Schema_4.Emp. The resolution of Cname to Schema_4.Company is a one-way join whereas the resolution of Cname to Schema_3.Branch is a two-way join - first between Schema_4.Emp and Schema_3.Industry and then between Schema_3.Industry and Schema_3.Branch. The former is more close than the latter.

In order to incorporate 'closeness', we define two terms - Distance and Weighted Count. Weighted count, WC, is a value associated with every query and it incorporates the 'density' principle. Distance incorporates the 'relationship' principle. These terms are explained below.

The relations present in a query resolution account for all the names in the user query. With each relation R that is present in the query resolution, we associate a 'reference count'. The reference count gives the number of times R has been referred to by the names in the user query.

Consider the following user query

```
SELECT Company Department
WHERE Cloc = 'Bangalore' OR Dname = 'Computer'
```
The names in the user query are Company, Department, Cloc and Dname. If names Company and Cloc refers to Schema_2.Comp and names Department and Dname refers to Schema_2.Dept, then \{Schema_2.Comp Schema_2.Dept\} is a query resolution as all the names in the user query are accounted for by the relations present in the query resolution. Also, the reference count for Schema_2.Comp and Schema_2.Dept is 2 as both account for two names in the user query.

WC is computed as follows:

Let the user query be of the form

```
SELECT name1, name2, ... namei
WHERE namei+1 = ' ' AND name_n = ' '
```

Here, the number of names in the user query is n. Let the relations which are referenced in the query resolution be R_1, R_2 ... R_m. Let their reference counts be C_1, C_2 ... C_m ordered in the descending order of reference counts. Then,

\[
\text{Weighted Count } WC = \sum_{i=1}^{m} (n - i + 1)C_i
\]

WC gives a measure of the density.

The Distance for a generated query is computed as follows:

\[
\text{Distance} = \text{number of joins} \times \text{number of schemas}
\]

Distance incorporates the 'relationship' principle.
4.2.4 Query Resolution Order

Applying the principles of density and relationship, the queries can be arranged in order of 'closeness' with the ones more close higher in the order. The resolutions, which have minimum number of relationships (joins), are higher in the order of 'closeness'. Within resolutions, which have the same number of joins, the ones in which density is maximized are more close than the others.

In terms of WC and Distance, this can be translated as follows:

1. First order the generated queries according to Distance starting with minimum Distance.
2. Within queries having equal Distance, order the queries in the descending order of Weighted Counts.

In the above ordering, the query with minimal Distance and maximum WC is the most 'close' query.

4.2.5 Redundant queries

Some query resolutions are redundant. In other words, they generate the same data as other queries. Consider, the following query

```
SELECT Company Department
WHERE Cloc = 'Bangalore' OR Dname = 'Computer'
```
Consider two generated queries.

```sql
FROM Schema_2.Comp, Schema_2.Dept
WHERE Schema_2.Comp.C# = Schema_2.Dept.C# AND
  (  
    Schema_2.Comp.Cloc = 'Bangalore' OR
    Schema_2.Dept.Dname = 'Computer'
  )
```

and

```sql
WHERE Schema_2.Comp.C# = Schema_2.Dept.C# AND
  (  
    Schema_2.Comp.Cloc = 'Bangalore' OR
    Schema_2.Dept.Dname = 'Computer'
  )
```

The data selected in the second query is identical to the data retrieved in the first query. This is because after taking the equi-join of Company and Industry, there is no additional data retrieved from Industry. The second query here is a redundant query. In our system the redundant queries are not generated.
4.2.6 An Example

Consider the following user query Q

\[
\text{SELECT Company Department} \\
\text{WHERE Cloc = 'Bangalore' OR Dname = 'Computer'}
\]

The aim is to find a query resolution that has minimum distance, and within this minimum Distance, has the maximum Weighted Count. The query resolution, at the same time, should also account for all the names in the user query i.e. atleast one of the relations in the query resolution should refer to it.

Consider the schema shown in Figure-14 of Chapter-3.

One way to resolve the names in the user query is by considering Schema_1. The relation Department in Schema_1 resolves for all the names in the user query.

In order to calculate the Distance, we need to consider the number of joins and the number of schemas. The number of schemas used is 1 and the number of join is 0.

Therefore, the Distance is

\[
\text{Distance} = \text{number of joins} \times \text{number of schemas} = 0
\]

In order to calculate the Weighted Count, we need to consider the reference counts of the relations that are referenced in the Query Resolution. In the formula

\[
WC = \sum_{i=1}^{m} (n - i + 1)Ci
\]
where

\( n = \) number of names in the user query = 4

\( m = \) number of referenced relations = 1

\( C_1 = \) reference count of the first referenced relation (Schema_1.Department) = 4

then

\[
WC = \sum_{i=1}^{n} (4 - i + 1)C_i = 16
\]

The corresponding generated query for the query resolution

\{Schema_1.Department\} is therefore

\[
\text{SELECT Schema_1.Department.D#, Schema_1.Department.Name,}\\
\text{Schema_1.Department.Comp, Schema_1.Department.Cloc}\\
\text{FROM Schema_1.Department}\\
\text{WHERE}\\
\text{(}\\
\text{Schema_1.Department.Cloc = 'Bangalore' OR}\\
\text{Schema_1.Department.Name = 'Computer'}\\
\text{)}\\
\]

It can be noted that the query resolution \{ Schema_1.Department \} account for all the names in the user query and therefore can be considered the most desired one from the user’s point of view. This query resolution is further used to prevent generating redundant query resolutions. That is, any query resolution, generated thereafter, that contains all the relations in this query resolution is considered redundant. These redundant query resolutions are not generated by our system.
Another way to resolve the names in the user query is by considering Schema_3. There are two relations in Schema_3 namely Industry and Department. Schema_3.Industry accounts for the three names in the user query i.e. Company, Department and Cloc whereas Schema_3.Branch accounts for name Dname.

The Distance is

\[ \text{Distance} = \text{number of joins} \times \text{number of schemas} = 1 \times 1 = 1 \]

The Weighted Count is calculated as follows:

\[ n = \text{number of names in the user query} = 4 \]
\[ m = \text{number of referenced relations} = 2 \]
\[ C_1 = \text{reference count of the first referenced relation (Schema_3.Industry)} = 3 \]
\[ C_2 = \text{reference count of the second referenced relation (Schema_3.Branch)} = 1 \]

\[ WC = \sum_{i=1}^{2} (4 - i + 1)C_i = 15 \]

The corresponding generated query for the query resolution is

```
AND
  (Schema_3.Industry.Cloc='Bangalore' OR
   Schema_3.Branch.Dname='Computer')
```
Schema_2 can also be used to resolve the names in the user query. The relation Comp accounts for the names Company and Cloc whereas the relation Dept accounts for the names Department and Dname.

The Distance is computed as follows

\[
\text{Distance} = \text{number of joins} \times \text{number of schemas} = 1 \times 1 = 1
\]

The Weighted Count is calculated as follows:

\[
n = \text{number of names in the user query} = 4
\]

\[
m = \text{number of referenced relations} = 2
\]

\[
C_1 = \text{reference count of the first referenced relation (Schema_3.Industry)} = 2
\]

\[
C_2 = \text{reference count of the second referenced relation (Schema_3.Branch)} = 2
\]

then

\[
WC = \sum_{i=1}^{2} (4 - i + 1)C_i = 14
\]

The corresponding generated query is

SELECT Schema_2.Comp.C#, Schema_2.Comp.Cname, Schema_2.Comp.Cloc,
to_char(Schema_2.Dept.Dno), Schema_2.Dept.Dname
FROM Schema_2.Comp, Schema_2.Dept
WHERE Schema_2.Comp.C#=Schema_2.Dept.C#
AND
(  
    Schema_2.Comp.Cloc='Bangalore' OR
    Schema_2.Dept.Dname='Computer'
  )
Although, the distance of the query resolutions \{\text{Schema}_3.\text{Industry} \text{Schema}_3.\text{Branch}\} and \{\text{Schema}_2.\text{Comp} \text{Schema}_2.\text{Dept}\} are same but the Weighted count of the former is greater than that of the latter. Therefore, the query ordering strategy, discussed before, will generate the query resolution \{\text{Schema}_3.\text{Industry} \text{Schema}_3.\text{Branch}\} before the query resolution \{\text{Schema}_2.\text{Comp} \text{Schema}_2.\text{Dept}\} as the former is more close than the latter.

Continuing in this way, SIQS generates 7 non-redundant query resolutions in a desired order. These query resolutions with their Distance(D) and Weighted Count(WC) are shown in Figure-3.

<table>
<thead>
<tr>
<th>No</th>
<th>Resolutions</th>
<th>D</th>
<th>WC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\text{Schema}_1.\text{Department}</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>\text{Schema}_3.\text{Industry} \text{Schema}_3.\text{Branch}</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>\text{Schema}_2.\text{Comp} \text{Schema}_2.\text{Dept}</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>\text{Schema}_2.\text{Dept} \text{Schema}_4.\text{Comp}</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>\text{Schema}_3.\text{Branch} \text{Schema}_4.\text{Company} \text{Schema}_4.\text{Emp}</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>\text{Schema}_2.\text{Dept} \text{Schema}_3.\text{Industry} \text{Schema}_4.\text{Emp}</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>\text{Schema}_2.\text{Comp} \text{Schema}_3.\text{Branch} \text{Schema}_4.\text{Emp}</td>
<td>6</td>
<td>13</td>
</tr>
</tbody>
</table>

Figure-3 – Query Resolution with Distance and Weighted Count

Each of the query resolutions shown in Figure-3 is transformed into a generated query. The corresponding generated queries, for the query resolutions shown in Figure-3, are shown in Figure-4.
**Generated Query**

1. 

```sql
FROM Schema_1.Department
WHERE (Schema_1.Department.Cloc='Bangalore' OR Schema_1.Department.Name='Computer')
```

2. 

```sql
```

3. 

```sql
FROM Schema_2.Comp, Schema_2.Dept
```

4. 

```sql
```

**Figure-4 – Generated Query**
AND to_char(Schema_4.Company.Cno)=to_char(Schema_4.Employee.Cno)
AND

WHERE Schema_2.Dept.C# = to_char(Schema_4.Emp.Cno)
AND

AND Schema_2.Comp.C# = to_char(Schema_4.Employee.Cno)
AND
4.3 Presenting the Result

As seen above, the generated queries involve different path expressions for the same name in different schemas. These different expressions will yield different results. For example, in the query

```sql
SELECT Department
```

the name Department has many mappings. It can be resolved to the relation Department in Schema_1 and relation Dept in Schema_2; as an attribute of relation Industry in Schema_3 and as an attribute of relation Emp in Schema_4. The four queries correspondingly will be

1) SELECT D#, Name, Comp, Cloc FROM Schema_1.Department
2) SELECT Dno, C#, Dname FROM Schema_2.Dept
3) SELECT Department FROM Schema_3.Industry
4) SELECT Department FROM Schema_4.Emp

The result in the first case is a relation with order 4, in the second case a relation of order 3 and a relation with a single attribute in the third and the fourth instances.

There are two ways in which the result can be presented to the user. In the first case, many results are presented to the user. These are as many as the number of different resolutions. In the above example, there are four results and of order 4, 3, 1 and 1 respectively.

The second method would be to present to the user just one set of tuples. The strategy adopted is as follows. For each set containing the result the missing
attributes are added to the set in such a way that the sets become union compatible. The result for these added attributes is set to null. The union of all the sets is performed. That is, the outer union of all the sets is taken. For example, the answer to the above query would have six attributes

D#, Dname, Company, Cloc, C#, Department

The attributes C# and Department would have null values wherever there is a value for the first four attributes and so on. This option, we feel, may not be the right way of presenting information to the user. For example, in the case when there is no common attribute among the sets, there would be nulls appearing in many fields of many tuples. This in effect is presenting the result of the first query followed by the second and so on. The resulting relation may not be a true reflection of the data.

We have therefore, chosen to present the different resolutions separately. That is, we have adopted the first option. In the above example there would be four results. Each result is presented to the user one after the other by the GUI interface.

4.3.1 Result Interface

The GUI presents the result to the user in the form of a window. The first window it presents is shown in Figure-5. The set of tuples with the attribute names is presented as a relation to the user. The user is given an option to either proceed to the next result by clicking the 'Next' button or quit by clicking the 'Quit' Button. The user can save the result, by clicking the 'Save' Button, for integration purpose. User can see the details of the result as well by clicking the 'Details' button. The user gets a new window on clicking the 'Details' button as shown in Figure-6.
Details show the pictorial representation of the schemas and their relations that participate to fetch the results. The joins are shown with the help of a line. The details also show the generated query of the corresponding result.

Figure-5 – Result Interface to show the result

Figure-6 – Result Interface to show Details of the result