# QUERY PROCESSING IN DYNAMIC FEDERATION FOR LOOSELY-COUPLED AND TIGHTLY-COUPLED SYSTEMS.

a dissertation submitted in partial fulfilment of the requirements for the M. Tech. (Computer Science) degree of the Indian Statistical Institute

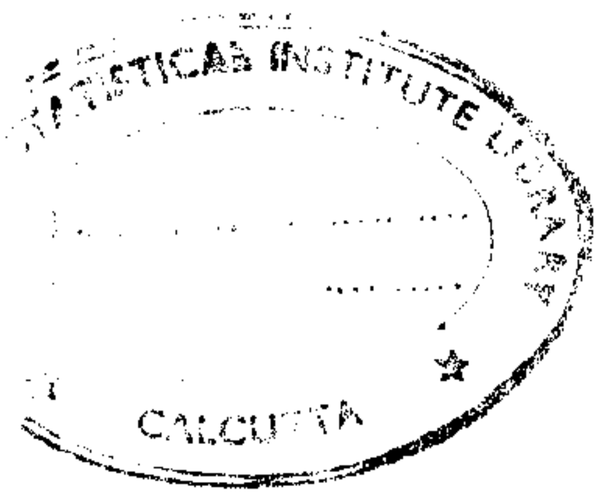
By

Rana Aich.

under the supervision of

Aditya Bagchi.
Computer & Statistical Service Center.
INDIAN STATISTICAL INSTITUTE
203, Barrackpore Trunk Road.
Calcutta-700035

July 26, 1996



**Indian Statistical Institute**
203, B.T. Road,
Calcutta- 700 035.

*Certificate of Approval*

This is to certify that the thesis titled, **QUERY PROCESSING IN DYNAMIC FEDERATION FOR LOOSELY-COUPLED AND TIGHTLY-COUPLED SYSTEMS.** submitted by **Rana Aich**, towards partial fulfilment of the requirements for the degree of M. Tech. in Computer Science at the Indian Statistical Institute, Calcutta, is an acceptable work for the award of degree.

Supervisor.
Computer & Statistical Service Center.
Indian Statistical Institute.
Calcutta-700 035.

External Examiner.

# Acknowledgements

The very first person whom I'd like to thank and whom I feel myself honoured to have worked with is my supervisor Prof. Aditya Bagchi, who is both an excellent teacher and a nice human being, and who has taught me a lot during my stay of two years.

I would like to thank all C.S.S.C. staffs for helping me to avail of the computing facilities and also offering their valuable advice, whenever I faced any problem.

I am also grateful to Mr. Sanjoy Bose and Mr. Tapan Kumar Adak for helping me with LaTeX, when I was writing this report.

Lastly, I would like to thank all my classmates, for making my two years' stint in ISI a memorable one.

Calcutta.
July 26, 1996

(Rana Aich.)

# Contents

# Chapter 1

# INTRODUCTION AND MOTIVATION

## 1.1 Introduction

A federated database system (FDBS) is a collection of autonomous databases where each member has local query processing facility for its own users and in addition, they can share certain global queries that are decomposed and distributed among the different members of the federation. Heimbigner & McLeod [1]considered a federated architechture as a loosely coupled structure which does not support interdatabase dependencies and does not have a global schema either. Sheth & Larson[2] on the other hand, described the federated system with much broader capabilities. Depending on the degree of autonomy present among the component databases, they classified FDBS as loosely-coupled and tighty-coupled systems. A tightly coupled system would definitely specify interdatabase relationships and would have a global schema. Larson et.al[5] identified the conflicts present among the participating databases. Besides identifying conflicts, Ozsu and Valduriez[3] also described the different stages of integration in a heterogeneous environment. However, all the proposals and research effort referred so far believe in the existence of all the component databases before the federation is built.

Bagchi[4] proposed a dynamic environment of Data Federation in which the component databases may join the federation or withdraw from it without affecting the transactions currently running. Component databases not involved in this dynamic change should continue to process their queries. While each component has its own local queries they should maintain a federation to model an organisational setup and thus would provide a structure for shared access. So a global query to the federated structure should be decomposed and distributed among the component databases. During processing, a participating database should not differentiate between a query local to itself and a sub-query assigned to it by a

decomposed global query. A dynamic federation demands that a federated structure should grow and shrink with addition or deletion of a database. Some applications need such a structure.

## 1.2 Motivation

We normally form a single database for an enterprise (An enterprise is a reasonably self-contained,commercial,scientific or any other organisation). In a relational system we can update the schema by adding new attributes to a relation or by forming a new relation according to the requirement. Update also includes deletion of attributes or relations.
The above considerations,which can be modelled by a single database, has the following limitations :
1. A large enterprise in general consists of a number of sub-enterprises. Though these sub-enterprises contain overlapping information(some relations are fully or partially common in the sense that they refer to same set of attributes), each such sub-enterprises may be large enough so as to be considered as a separate database.
2. Each sub-enterprise has users who are interested only with information related to that sub-enterprise and may not be interested in others.
3. Each database may need local autonomy for efficient maintenance and query processing.
4. Sub-enterprises should act independent to each other, so that error/crash in one should not hinder the local operations in other sub-enterprises.

If we consider a single database for the whole organisation, we cannot provide such facilities.
Let's consider a specific example :
A 'Factory' maintains a database that includes its accounts,stores, personnel and other departments. Later the factory management decides to make a 'Housing Complex' near the factory. This would involve different types of quarters, arrangement of necessary facilities like water supply, electric supply etc. and would ultimately demand a separate database. However, since the persons occupying the quarters in the 'Housing Complex' are employees in the factory, there may be some queries which would involve relations and attributes of both databases. So the concerned query need to be decomposed to generate local sub-queries which would then be distributed among the component databases. Similarly a databasae named 'Hospital' is added to this federation if the authority decides to open a Hospital to render efficient medical service to its employees. In the same manner, if a school is opened by the factory management for its employees, it should also be integrated with the Federation. However, if a public school is opened near the factory and most of the workers decide to

send their children to that school, the factory management may close its own school in the campus causing the withdrawal of the corresponding schema. This would effectively cause the federated structure to shrink.

# Chapter 2

# THE DYNAMIC FEDERATION

Bagchi[4] in his paper, proposed a dynamic federated structure, DYNAFED. It has been developed as a homogeneous system where all the component databases are relational. Since standard mapping rules are readily available for conversion from other data models to relational system, DYNAFED can as well be applied in heterogeneous environment. A subsystem for schema translation has to be placed as a front end to the proposed model for this purpose. The experimental system has been simulated as a centralised one under VAX/VMS environment. The component databases have been implemented using RDB/VMS. Both the loosely-coupled and tightly coupled system have been developed. In loosely-coupled one the participating databases can function quite independently while in tightly coupled system there are strong dependencies among the databases. The control of the federation is done by a data dictionary with a synonym table for the attributes of the participating databases. The entry to or exit from the federation by the component databases is order independent in case of loosely-coupled system. Query once decomposed and distributed as subquries,they are taken care of by the component databases participating in the processing. A SQL like language is used to specify the global query. After decomposition the component queries are presented in standard SQL to the corresponding databases. For any change in the participating schema the federated structure is also updated. Insertion, deletion, or modification of data in any component database is done independently for a loosely-coupled system. The federated structure has no direct access to any data present in any of the component databases. Thus it may be considered as a 'federated view' over the component relational schemas.
Tightly-coupled system mainly incorporates the generalisation/specialisation hierarchy. The semantic conflict, as mentioned in earlier proposals [3, 4, 5], has been resolved in this system. If an attribute/relation in database A is a specialisation of an attribute/relation present in database B, then B is a superschema of A or A is a subschema of B. This is similar to superclass subclass relationship in an object-oriented system.
So in the example of section 1.2, the relation "Doctor" in the "Hospital" schema is the

specialisation of relation "Employee" in the "Factory" schema. Similarly the attribute "Doctor_name" in "Doctor" is the specialisation of "Employee_name" in "Employee".
For mutually independent databases, the entry to and exit from the federation are still order independent. However, a superschema should join the federation before any of its subschema and cannot withdraw from it so long any of its subschema is present in the federation. So the tightly coupled-system conceptualizes a global schema. Here, the insertion, deletion, and modification of data for any component database have to be done through this global schema or the federated structure.

## 2.1 Loosely-coupled System :

### 2.1.1 System Structure :

The system has three basic interactive software modules - Creator, Controller, and Navigator. Whenever a new schema is defined, Creator accepts the relevant information through a user interface and generates the corresponding SQL file to create the database. Controller, on the other hand controls the creation and updation of federated structure. Navigator is the third module that helps to navigate through the federated structure. It is primarily responsible for decomposition of global queries and distribution of the component queries.
Let's consider two databases D1 and D2 with the following relations and attributes :

Database : $D1$

Relations

$$: R11 = (A, B, C, D)$$

$$: R12 = (A, D, E, F)$$

Database : $D2$

Relations

$$: R21 = (P, Q, R, S)$$

$$: R22 = (X, Y, R, B)$$

*Creation*

Creator provides a user interface for defining the relations, attributes, domains, constraints

etc. and creates the corresponding schemas.

Controller generates a Global Attribute Table (GAT). GAT is basically a synonym table where against each global name a list of synonyms are stored. Some of the interschema conflicts are also resolved through GAT. A row of this table has a set of synonyms represented by a global name. A global name may be supplied by the user or when the first synonym of a row is inserted its name is taken as global name by default. In case a default global name becomes a homonym to an existing one, user would have to supply the alternate name. From the second database onwards Creator after creating the schema asks for synonyms and homonyms. Synonyms are appropriately placed in GAT. Unless global name conversion is necessary homonyms are also placed in the table. 'Naming conflict' is resolved before placing a name in GAT. To explain the situation, let's consider that the two databases $D1$ and $D2$ are inserted one after the other. User provides the following information to the Creator.

Synonyms

$$: Q\ IN\ R21\ =\ E\ IN\ R12;$$

$$: Y\ IN\ R22\ =\ C\ IN\ R11;$$

Homonyms

$$: B\ IN\ R22\ \neq\ B\ IN\ R11;$$

$$GLOBAL\ NAME\ FOR\ R22.B\ IS\ B1;$$

After the insertion of $D1$ and $D2$ the condition of GAT is shown in the Figure below:

| Global Name | Parameter | Synonym1 | Parameter | Synonym2 | Parameter |
|---|---|---|---|---|---|
| A | | $D1.R11.A$ | | $D1.R12.A$ | |
| B | | $D1.R11.B$ | | | |
| C | | $D1.R11.C$ | | $D2.R22.Y$ | |
| D | | $D1.R11.D$ | | $D1.R12.D$ | |
| E | | $D1.R12.E$ | | $D2.R21.Q$ | |
| F | | $D1.R12.F$ | | | |
| P | | $D2.R21.P$ | | | |
| R | | $D2.R21.R$ | | $D2.R22.R$ | |
| S | | $D2.R21.S$ | | | |
| X | | $D2.R22.X$ | | | |
| B1 | | $D2.R22.B$ | | | |

Figure1 : Global Attribute Table after insertion of $D1$ and $D2$

Each synonym is associated with value constraints as parameters. Constraint type is specified for each synonym. It can be 'no constraint', 'set type constraint', 'null constraint' or 'range type constraint'. Constraint type is followed by reference to constraint-file which provides the detail.

*Conflicts resolution:*

1) Naming Conflict : To resolve the naming conflict each attribute name has been made unique by prefixing the corresponding database and relation names as shown in Figure. Naming conflict also occurs when a new attribute name conflicts against an existing global name. This is resolved through user's intervention. In the example, second appearence of B has changed to $B1$ by the user.

2) Domain-Unit conflict : It is needless to say that all the synonyms of an attribute must have the same domain. So domain information is associated with the global name as a parameter. However, the different synonyms may not have the same unit. Two attributes may represent temperature but one in Centigrade and the other in Fahrenheit. The global name, however, must be associated with one unit. Normally, the unit of the first synonym is assigned as the unit of the global name. Each synonym is also associated with its own unit. Since a global query is done by using the global attributes only, if a synonym has a unit different from the corresponding global one, conversion is necessary. So the parameters associated with each attribute include its unit, a bit to indicate whether conversion is needed and then a reference to the function for conversion.

3) Heterogeneity Conflict : Since we are designing a homogeneous system and all the component databases are relational i.e based on same data model - heterogeneity conflict is not present here.

4) Semantic Conflict : The set of values taken by an attribute in a databae may be included within the set of values taken by another attribute in a different database. For example, the values associated with 'Graduate-Student' will always be included in the general attribute 'Student'. In a semantic data model this conflict is resolved by generalisation/specialisation hierarchy. In AI domain it is more popularly known as IS-A relationship. In loosely-coupled environment this conflict does not arise since all the participating databases are absolutely autonomous and no interschema dependencies are considered. Here the attributes 'Graduate-student' and 'Student' would appear as synonym without considering the inclusion dependency. This confilct is however considered in a tightly coupled system.

5) Structural Conflict : In a heterogeneous environment this conflict becomes extremely important during schema intregation. The extent of conflict depends on how many ways a reality can be modeled in different component databases. Semantic and object-oriented models can support many different but equivalent representations of the same reality. In case of relational model, however this problem is not so apparent in both cases of loosely-coupled and tightly-coupled system.

*Alteration :*

Any change in the participating schema can be specified through the Creator again and it generates the corresponding set of SQL commands and incorporates the change. However, various types of changes may have to be done in GAT.

1) If a new attribute is added, it can either be a synonym to an existing attribute and will be placed in the proper position or it will give rise to a new global attribute.

2) When a new attribute is added or the unit of an existing attribute is changed,unit conversion may become necessary. New function may also have to be added to the heap of functions and proper references have to be established.

3) When the domain of an attribute is changed, a synonym may become a homonym to an attribute. So the changed attribute will be delinked from its existing position and it will either be added as a synonym to some other attribute or it will give rise to a new global name.

4) If the value constrint for an attribute is changed, its constraint type parameter may have to be changed and new addition may have to be done to the constraint-file establishing new references.

*Deletion :*

Deletion can be of three types :

1) In case of attribute deletion, the concerned attribute is delinked from the global attribute table. If the deleted attribute is the first member of a set of synonyms and its name has been used as the default global name, the user may alter global name or may continue with it.

2) If a relation is deleted, all its attributes are deleted.

3) If a database is deleted, all its relations and their attributes are deleted.

The word deletion may create a confusion here. Deletion of a database from the federation does not necessarily mean that the database is actually deleted. After it is delinked from the federation, a database may still maintain its seperate and independent existence. In case a database is really to be removed, it has to be done locally. However, if a user tries to delete a database locally without delinking it from the federation, the security system would prevent him.

*Insertion, Deletion & Updation of data :*

Since all the component databases are absolutely autonomous insertion, deletion or updation of data in each database is done independently without involving the federated structure or any other database participating in the federation.

### 2.1.2 Query Processing in loosely-coupled System :

Query Processing in the federated database can be divided into global query processing and local query processing. The queries are presented to the system in the following form :

Select (... *attribute_list* ...)
where (... *predicate_expression* ...)

where attributes in the < *attribute_list* > and in the < *predicate_expression* > are the global names from GAT. Any synonym from any database can take part in the query processing provided the database can satisfy the predicate expression present in the query. Predicate expression contains boolean/relational operators. From the query entered by the user, all the attributes are extracted. Now the list of databases which covers the attributes are selected. The algorithm developed for this purpose tries to cover the required set of attributes with minimum number of databases. Since this is an well-known NP-complete problem a greedy algorithm has been suggested as a sub-optimal solution.

**Query decomposition and distribution algorithm :-**

**step-1 :** All the attributes appearing in the 'select' and 'where' clause are considered. Only the distinct set is taken and the attribute appearing more than once would be discarded.

**step-2 :** The above distinct set of attributes is checked whether all of them are global attributes or not. Only global attribute has to be considered (as the user has to give the query using global attributes only).

**step-3 :** For each selected global name, all the databases that cover the attribute are found from GAT.

**step-4 :** Against each database name, the global attribute names covered by it is listed. The databases are then arranged in the descending order of their cardinality of coverage. So the database that covers the maximum number of attributes present in the query would be at the top of the list.

**step-5 :** Using the above list the minimum cover is found out.

The necessary algorithm is given below :

**QUERY DECOMPOSITION ALGORITM :**

INPUT : set global attributes present in the user query;

OUTPUT: set of database names participating in the processing the query alongwith the list of attributes covered by each of them;

{

**step 1:**

$S$ = { All the attributes present in the query i.e attributes present in the **SELECT** clause as well as in the **WHERE** clause }

**step 2:**

From the data dictionary,

$Sd[j]$ = { All the attributes present in the database $D[j]$ }

/* corresponding global attribute names are included in the set*/

**step 3:**

for $j$ = 1 *to* $k$ /* k being the number of databases present in the federation */

{

$s[j] = S \cap Sd[j]$;

/* the attributes in S that are covered by the database $D[j]$ */

$n[j] = cardinality\{s[j]\}$;

/*the number of attributes present in $s[j]$ */

}

**step 4:**

sort $n[j]$ in the descending order and also sort the corresponding $s[j]$ in the same order;

So, in this list,

if the first member be $s[1]$ then $n[1]$ has the highest cardinality value i.e $s[1]$ or the corresponding database $D[1]$ covers the maximum number of attributes present in S;

**step 5:**

select $s[1]$ the database covering maximum number of attributes;

TOTAL = $Sd[1]$; /* union of the attributes present in the selected database */

RESULT = $\{s[1]\}$; /* set of sets that provides the list of the selected

database alongwith the list of global attributes covered by each of them
*/
if (S - RESULT $=_{\overline{7}}$ $\phi$) break;
for $i$ = 2 *to* $k$

{
if ($Sd[i] \cap$ TOTAL != $\phi$)
{
$TOTAL = TOTAL \cup Sd[i]$;
$RESULT = RESULT \cup \{s[i]\}$;
} /* end if */

if(S - RESULT == $\phi$) break;
} /* end for */

}

Thus, for each global query the decomposition algorithm provides a list of database names alongwith their associated attributes.

**SUB-QUERY GENERATION & DISTRIBUTION ALGORITHM :**

INPUT : RESULT as obtained from QUERY DECOMPOSITION ALGORITHM;
OUTPUT : Sub-queries to selected database;
For each member of RESULT,

**step : 1** extract corresponding database name;
convert each global attribute covered by it to the local attribute name using GAT;

**step : 2** with the help of original user's query and the list of attributes
obtained in **step 1**,
identify the attributes which should be placed under **SELECT** clause and
those to be placed in **WHERE** clause;
predicate expressions to be added to the **WHERE** clause are obtained from
the original query;

**step 3 :** form the proper SQL query for the selected database;
continue till RESULT is exhausted;

end of algorithm;

The selection of relations within a database has not been considered here. That would need another algorithm similar to the DECOMPOSITION ALGORITHM considered earlier. This dissertation, however, considered universal relations for each database and avoided the formation of **FROM** clause in the SQL query.

All the intermediate relations generated by the sub-queries would send their results to a central point for the navigator to consolidate. So the space requirement at the central point would be quite high. This large space requirement may possibly be avoided to some extent by adopting some optimizing strategies. The intermediate relations may be transferred selectively from one component database to another and necessary semijoins may be done. Tradeoffs need to be calculated between data transfer and multiple semi join costs on one side and the cost for large space requirement at the central point on the other. Query optimisation is yet to be considered.

# Chapter 3

# TIGHTLY-COUPLED SYSTEM :

## 3.1 An overview of Tightly-Coupled System :

In this dissertation work we were interested mainly in converting a loosely-coupled system into tightly coupled one. At the first stage of such conversion, inclusion dependencies have been considered.

In the 'Factory-Housing-School-Hospital' federation discussed earlier let's consider,
Database 'Factory' has a relation,
employee = (*employee_no*, *name*, *address*, *sal*, *date_of_joining*)
Database 'Hospital' has a relation,
doctor = (*doctor_no*, *name*, *address*, *sal*, *data_of_joining*, *splztion*)
Database 'School' has a relation,
teacher = (*teacher_no*, *name*, *address*, *sal*, *date_of_joining*, *subject*)

At the time of creation, we can define inclusion dependencies as :
between relations,
doctor IS-A employee
teacher IS-A employee
and between attributes,
doctor_no IS-A employee_no
teacher_no IS-A employee_no
etc.

These IS-A relations would give rise to a different set of considerations not present in the loosely-coupled system.

1) The 'employee' relation should keep information about all the employees including doctors and teachers. Logically the 'doctor' and 'teacher' relations should keep only the special attributes 'splztion' and 'subject' respectively and other attributes should be obtained by inheritance. On the other hand, similar to loosely-coupled system a local query to the 'Hospital' or 'School' database should be answered locally. So should some data be repeated between 'Doctor' and 'Employee' or 'Teacher' and 'Employee'.

2) Attributes 'Factory.employee.name','Hospital.doctor.name' and 'School.teacher.name' would still appear as synonyms in GAT. However for inclusion dependencies local queries cannot be served locally always. For example, 'name' information should not be available at the 'Hospital' and 'School' databases, but should be obtained by inheritance from the 'Factory' database. At the time of schema integration 'Hospital.doctor.name' and 'School.teacher.name' should be marked as dummy attributes by a special bit associated with each synonym in GAT. This has been named as 'dependent_bit'. So, even a local query has to be processed through the federation and may involve more than one database. This coupling by inclusion dependencies would definitely ask for new set of algorithms for the Navigator.

3) Here, 'Factory' is the 'super schema' and 'Hospital' and 'School' are the 'sub schemas'. Insertion of databases to the federation is no longer order independent.

4) Insertion, deletion, or modification of data in a component database can no longer be independent of federation. So new algorithms need to be developed. The query language for the federated database has to be augmented as well.

## 3.2 Query processing in tightly-coupled system :

In tightly-coupled system the query processing is almost similar to that in the loosely-coupled one except for the fact that special attention has to be given for inclusion dependencies. Here again we consider the queries in the following form :

Select (...attribute_list...)
where (...predicate_expression..)

Since datavalues for all the inherited attributes are residing in the superschema, a query processing algorithm needs to identify such databases even when a local query for some other database is considered. For example, data for attributes doctor_name, address, salary and date_of_joining in the 'Hospital' database can only be found in the 'Factory' database. The

dependent_bit for these attributes are all zeroes in the 'Hospital' database whereas corresponding values in the 'Factory' database are all one. The primary key values, however, are repeated in all the relevant databases. So the dependent_bit in GAT is one for all its synonyms. The primary key is used for linking the corresponding tuples in different databases during inheritence. For example, "employee_no" and "teacher_no" are the primary keys and their values are repeated in "School" schema even when they are available in the superschema "Factory". Without this primary key a "teacher" with certain "subject" in the "School" database cannot be linked with the corresponding "employee" tuple in the "Factory" database.

**QUERY DECOMPOSITION ALGORITHM IN A TIGHTLY-COUPLED SYSTEM :**

**step 1:** All the attributes appearing in the **SELECT** clause as well as **WHERE** clause are considered. Only the distinct set is taken and same attribute appearing more than once is discarded.

**step 2:** The above distinct set of attributes is checked whether all of them are global attributes or not. ( The user has to give the query using global attributes only).

**step 3:** For each selected global name all the databases that cover the attribute are found from GAT.

**step 4:** For each database name all the attributes covered by it is listed and sorted in the descending order of their cardinality of coverage. So the database that covers the maximum number of attributes present in the query would be at the top of the list.

**step 5:** Start with the first database of the list. It covers the maximum number of attributes present in the query. Find among those attributes covered by the database the one which is the primary key.

**step 6:** If the primary key is found then find for each attribute covered by the database the status of the dependent_bit.
If the dependent_bit is one then include the attribute in the subquery.
Else search the GAT to find a synonym for which the dependent_bit is one . The corresponding database actually contains the data.

**step 7:** If the primary key is not found in the set of attributes covered by the database under consideration, try to find from GAT the primary key for the database and add it to the subquery. For the other attributes find the status of dependent_bit and act accordingly as described in **step 5**.

The necessary algorithm is given below :

**ALGORITM :**

INPUT : set of global attributes present in the user's query;

OUTPUT: set of database names participating in the processing of query along-with the list of attributes covered by each of them;

{

**step 1:**

S = { All the attributes present in the query i.e attributes present in the **SELECT** clause as well as in the **WHERE** clause }

**step 2:**

From the data dictionary,

$Sd[j]$ = { All the attributes present in the database $D[j]$ }

/* corresponding global attribute names are included in the set */

**step 3:**

For j = 1 to k /* k being the number of databases present in the federation */

{

$s[j] = S \cap Sd[j]$; /* the set of attributes in S that are covered by the database $D[j]$ */

$n[j] = cardinality\{s[j]\}$; /* the number of attributes present in $s[j]$ */

}

**step 4:**

sort $n[j]$ in the descending order and also sort the corresponding $s[j]$ in the same order;

So in this list, if the first member be $s[1]$ then $n[1]$ has the highest cardinality value, i.e. $s[1]$ or the corresponding database $D[1]$ covers the maximum number of attributes present in S;

**step 5:**

TOTAL = $Sd[1]$; /* union of the attributes present in the selected database */

RESULT = $\{s[1]\}$; /* a set of sets that provides the list of the selected databases alongwith the list of global attributes covered by each of them

```
*/

for j = 1 to k

    {
    if( Sd[j] ∩ TOTAL != φ)
    {
    TOTAL = TOTAL ∪ Sd[j];
    RESULT = RESULT ∪ {s[j]};
    } /* end if */
    for each attribute in s[j],
        {
        if(the dependent_bit in GAT == 0)
        {
        find the synonym in GAT for which the dependent bit is 1;
        find the corresponding database D[*] and the sets s[*]
        and Sd[*];
        TOTAL = TOTAL ∪ Sd[*];
        RESULT = RESULT ∪ {s[*]};
        } /* end if */
    } /* end for */
    if(s[j] contains no primary key) {
    find the corresponding primary key in GAT; /* for primary
    key, dependent_bit in all synonyms will be 1 */
    find the corresponding database D[*] and the sets s[*] and
    Sd[*];
    TOTAL = TOTAL ∪ Sd[*];
    RESULT = RESULT ∪ {s[*]};
    } /* end if */
    if( S - RESULT == φ) break;
            } /* end for */

} /* end */
```

# Chapter 4

# IMPLEMENTATION

The program developed takes a new incoming database as input and integrates it into the existing multidatabase. The new database is formed in an interactive session with the user. The program form an SQL file which when executed produces the database. After the new database is formed, conceptual integration begins. If the database is entered as a first case, then no integretion is to be done and it is directly entered into the multidatabase.

The program which has been developed for the federated structure deals with the following concepts :

1. **Data Dictionary.**
2. **Global Attribute Table.**

When the user enters the input for each database in an interactive session, the following files are to be updated to maintain the data dictionary :

1. **Database File.**
2. **Relation File.**
3. **Attribute File.**

The following files are to be updated for the integration process and for serving the global as well as local queries :

1. **Global File.**
2. **Synonym File.**

These two file together provides the concept of "Global Attribute Table".

File containing the SQL commands that define the new database is also created. A global command procedure as available in VAX/VMS environment does all the necessary updation of files and creation of new database.

**Data Dictionary :** Data dictionary contains following three files :

*** Database File :** Each record of the database file contains the following fields :

1. Valid_bit : To check whether a particular database is valid or not i.e it has been deleted or not.

2. Busy_bit : To check whether the particular database is participating in processing the current transaction or not.

3. Database_name : The name of the database.

4. Pointer to relation : Keeps the address of the relation present in the database. Relations are available in Relation_file.

*** Relation File :** Each record of the Relation File contains the following fields :

1. Valid_bit : To check whether a particular relation is valid or not i.e. it has been deleted or not.

2. Relation_name : the name of the relation.

3. Pointer to attribute : This field of relation file contains the address of the first attribute of this relation.

4. Pointer to next relation : Since a database can have more than one relation, this field keeps the address of the next relation under the same database.

*** Attribute File :** Each record of the attribute file contains the following fields :

1. Valid_bit : To check whether a particular attribute is valid or not i.e. it has been deleted or not.

2. Attribute_name : The name of the attribute.

3. Domain : It provides the domain (integer,real etc.) of an attribute.

4. Pointer to the next attribute : This field keeps the address of the next attribute under the same relation.

5. Primary_key_bit : This bit is 1 if the attribute is a primary key, otherwise it is 0.

**Global Attribute Table :** Global attribute table is basically a synonym table, where against each global name a list of synonyms are stored. The program which has been developed uses two files for the above purpose :

*** Global File :** Each record of the global file contains the following fields:

1. Unit : This is the unit of global_attribute.

2. Attribute_name : The name of global_attribute.

3. Domain : The domain of global_attribute.

4. First_synonym_address : It keeps the address of the first synonym of a global attribute.

5. Primary_key_bit : This bit is 1 if the attribute is a primary key, otherwise it is 0.

*** Synonym File :** Each record of the synonym file contains the following fields :

1. Valid_bit : Whether the synonym name is valid or not.

2. Unit : This is the unit of the synonym.

3. Conversion_bit : If the unit of an attribute in the synonym file differs from that in the global file, then this bit indicates that conversion of unit is necessaery. A file containing the conversion routines for such such conversions at different cases would be necessary. Proper reference would be provided from the synonym file. We have not implemented this feature.

4. Domain : This expresses the domain of the synonym.

5. Attribute_name : The name of the attribute.

6. Relation_name : The name of the relation under which the attribute falls.

7. Database_name : The name of the participating database containing the synonym.
8. Dependent_bit : If the data is stored under this attribute column then dependent_bit is one. Otherwise it is zero.

9. Primary_key_bit : This bit is 1 if the attribute is the primary_key, otherwise it is 0.

10. Constraint_type : The constraint type is 0 if it is of set type.
Example :- Set type constraints : sex is (M,F).
: day is (sun,mon,.....,sat).
The constraint type is 1 if it is of range type.
Example :- (Salary >= 10000 & salary <= 4500)
The constraint type is 2 if it is of null type.

11. Next_synonym_address : This field keeps the address of the next synonym. If no more synonym is there then -1 is placed.

12. Previous_synonym_name : The name of the attribute that is synonym to the present attribute is kept in this field. For the attribute contained by the first database, this field is the attribute name itself.

**How the Data_Dictionary and GAT is updated and maintained :**

Whenever the user enters the first database with its relations and attributes, all the files of the data dictionary are updated. For GAT the first attribute name and other relevant information are stored in the global file. If the same attribute name belonging to different relation is entered again, it would be considered as a synonym and its name would be prefixed by the appropriate relation and database_name to make it unique.

From the second database onwards when a new attribute is inserted, it is declared whether the new one is synonym or homonym to an existing global attribute. Accordingly the new attribute would be placed in the Global Attribute Table (GAT). A homonym to an existing global attribute or an absolutely new attribute would give rise to a new global name.

# Chapter 4

# RESULTS

**FILES OF DATA DICTIONARY :**

---

**DATABASE FILE :**

1, *factory*, 0
2, *school*, 96
3, *hospital*, 160;

**RELATION FILE :**

*employee*, 0, 32
*department*, 270, 64
*dependent*, 486, −1
*teacher*, 648, 128
*student*, 918, −1
*doctor*, 1188, 192
*patient*, 1458, −1;

**ATTRIBUTE FILE :**

0, *ID_DOM*, *emp_no*, 2, 1, 2, 54
0, *CH_DOM*, *emp_name*, 2, 0, 2, 108
0, *CH_DOM*, *address*, 2, 0, 2, 162

0, *FL_DOM*, *basic*, 1, 0, 2, 216
0, *ID_DOM*, *dept_no*, 2, 0, 2, −1
0, *ID_DOM*, *dept_no*, 2, 1, 2, 324
0, *CH_DOM*, *dept_name*, 2, 0, 2, 378
0, *CH_DOM*, *manager*, 2, 0, 2, 432
0, *FL_DOM*, *budget*, 1, 0, 2, −1
0, *CH_DOM*, *dep_name*, 2, 1, 2, 540
0, *ID_DOM*, *emp_no*, 2, 0, 2, 594
0, *CH_DOM*, *relation*, 2, 0, 2, −1
0, *ID_DOM*, *t_no*, 2, 1, 2, 702
0, *CH_DOM*, *t_name*, 2, 0, 2, 756
0, *CH_DOM*, *t_address*, 2, 0, 2, 810
0, *FL_DOM*, *t_basic*, 1, 0, 2, 864
0, *CH_DOM*, *subject*, 2, 0, 2, −1
0, *ID_DOM*, *roll_no*, 2, 1, 2, 972
0, *ID_DOM*, *class*, 2, 1, 2, 1026
0, *ID_DOM*, *std_age*, 2, 0, 2, 1080
0, *CH_DOM*, *std_name*, 2, 0, 2, 1134
0, *ID_DOM*, *guar_no*, 2, 0, 2, −1
0, *ID_DOM*, *doc_no*, 2, 1, 2, 1242
0, *CH_DOM*, *doc_name*, 2, 0, 2, 1296
0, *CH_DOM*, *doc_address*, 2, 0, 2, 1350
0, *FL_DOM*, *doc_basic*, 1, 0, 2, 1404
0, *CH_DOM*, *specialization*, 2, 0, 2, −1
0, *ID_DOM*, *p_no*, 2, 1, 2, 1512
0, *CH_DOM*, *p_name*, 2, 0, 2, 1566
0, *CH_DOM*, *p_address*, 2, 0, 2, 1620
0, *ID_DOM*, *doc_no*, 2, 0, 2, −1;

**FILES OF GLOBAL ATTRIBUTE TABLE :**

---

**GLOBAL FILE :**

0, *emp_no*, *ID_DOM*, 1, 2, 0
0, *emp_name*, *CH_DOM*, 0, 2, 102
0, *address*, *CH_DOM*, 0, 2, 204
0, *basic*, *FL_DOM*, 0, 1, 306

0, *dept_no*, *ID_DOM*, 0, 2, 408
0, *dept_name*, *CH_DOM*, 0, 2, 612
0, *manager*, *CH_DOM*, 0, 2, 714
0, *budget*, *FL_DOM*, 0, 1, 816
0, *dep_name*, *CH_DOM*, 1, 2, 918
0, *relation*, *CH_DOM*, 0, 2, 1122
0, *subject*, *CH_DOM*, 0, 2, 1632
0, *roll_no*, *ID_DOM*, 1, 2, 1734
0, *class*, *ID_DOM*, 1, 2, 1836
0, *std_age*, *ID_DOM*, 0, 2, 1938
0, *specialization*, *CH_DOM*, 0, 2, 2652
0, *p_no*, *ID_DOM*, 1, 2, 2754
0, *p_name*, *CH_DOM*, 0, 2, 2856
0, *p_address*, *CH_DOM*, 0, 2, 2958;

**SYNONYM FILE :**

*factory*, *employee*, *emp_no*, 0, 0, 2, 1, 1, 52, 1020, *emp_no*
*factory*, *employee*, *emp_name*, 0, 0, 2, 1, 0, 52, 1326, *emp_name*
*factory*, *employee*, *address*, 0, 0, 2, 1, 0, 52, 1428, *address*
*factory*, *employee*, *basic*, 0, 0, 2, 1, 0, 52, 1530, *basic*
*factory*, *employee*, *dept_no*, 0, 0, 2, 1, 0, 52, 510, *dept_no*
*factory*, *department*, *dept_no*, 0, 0, 2, 1, 1, 52, −1, *dept_no*
*factory*, *department*, *dept_name*, 0, 0, 2, 1, 0, 52, −1, *dept_name*
*factory*, *department*, *manager*, 0, 0, 2, 1, 0, 52, −1, *manager*
*factory*, *department*, *budget*, 0, 0, 2, 1, 0, 52, −1, *budget*
*factory*, *dependent*, *dep_name*, 0, 0, 2, 1, 1, 52, 2040, *dep_name*
*factory*, *dependent*, *emp_no*, 0, 0, 2, 1, 0, 52, 1224, *emp_no*
*factory*, *dependent*, *relation*, 0, 0, 2, 1, 0, 52, −1, *relation*
*school*, *teacher*, *t_no*, 0, 0, 2, 1, 1, 52, 2142, *emp_no*
*school*, *teacher*, *t_name*, 0, 0, 2, 0, 0, 52, 2346, *emp_name*
*school*, *teacher*, *t_address*, 0, 0, 2, 0, 0, 52, 2448, *address*
*school*, *teacher*, *t_basic*, 0, 0, 2, 0, 0, 52, 2550, *basic*
*school*, *teacher*, *subject*, 0, 0, 2, 1, 0, 52, −1, *
*school*, *student*, *roll_no*, 0, 0, 2, 1, 1, 52, −1, *
*school*, *student*, *class*, 0, 0, 2, 0, 1, 52, −1, *
*school*, *student*, *std_age*, 0, 0, 2, 1, 0, 52, −1, *
*school*, *student*, *std_name*, 0, 0, 2, 0, 0, 52, −1, *dep_name*
*school*, *student*, *guar_no*, 0, 0, 2, 0, 0, 52, 2244, *emp_no*

*hospital*, *doctor*, *doc_no*, 0, 0, 2, 1, 1, 52, 3060, *emp_no*

*hospital*, *doctor*, *doc_name*, 0, 0, 2, 0, 0, 52, −1, *emp_name*

*hospital*, *doctor*, *doc_address*, 0, 0, 2, 0, 0, 52, −1, *address*

*hospital*, *doctor*, *doc_basic*, 0, 0, 2, 0, 0, 52, −1, *basic*

*hospital*, *doctor*, *specialization*, 0, 0, 2, 1, 0, 52, −1, ∗

*hospital*, *patient*, *p_no*, 0, 0, 2, 1, 1, 52, −1, ∗

*hospital*, *patient*, *p_name*, 0, 0, 2, 1, 0, 52, −1, ∗

*hospital*, *patient*, *p_address*, 0, 0, 2, 1, 0, 52, −1, ∗

*hospital*, *patient*, *doc_no*, 0, 0, 2, 0, 0, 52, −1, *emp_no*;

**SQL FILE :**

```
CREATE SCHEMA FILENAME factory;

CREATEDOMAINCH_DOMCHAR(20);
CREATEDOMAINV_CH_DOMVARCHAR(20);
CREATE DOMAIN TI_DOM TINYINT;
CREATE DOMAIN ID_DOM INTEGER;
CREATE DOMAIN RE_DOM REAL;
CREATE DOMAIN FL_DOM FLOAT;
CREATE DOMAIN DATE_DOM DATE;
CREATE TABLE employee (
emp_no ID_DOM,
emp_name CH_DOM,
address CH_DOM,
basic FL_DOM,
dept_no ID_DOM
primary key(emp_no)
);
CREATE TABLE department (
dept_no ID_DOM,
dept_name CH_DOM,
manager CH_DOM,
budget FL_DOM
primary key(dept_no)
);
CREATE TABLE dependent (
dep_name CH_DOM,
emp_no ID_DOM,
relation CH_DOM
primary key(dep_name)
);
COMMIT

FINISH
CREATE SCHEMA FILENAME school;
```

*CREATEDOMAINCH_DOMCHAR*(20);
*CREATEDOMAINV_CH_DOMVARCHAR*(20);
CREATE DOMAIN TI_DOM TINYINT;
CREATE DOMAIN ID_DOM INTEGER;
CREATE DOMAIN RE_DOM REAL;
CREATE DOMAIN FL_DOM FLOAT;
CREATE DOMAIN DATE_DOM DATE;
CREATE TABLE teacher (

t_no ID_DOM,
t_name CH_DOM,
t_address CH_DOM,
t_basic FL_DOM,
subject CH_DOM
primary key(t_no)
);
CREATE TABLE student (

roll_no ID_DOM,
class ID_DOM,
std_age ID_DOM,
std_name CH_DOM,
guar_no ID_DOM
primary key(roll_no)
primary key(class)
);
COMMIT

FINISH
CREATE SCHEMA FILENAME hospital;

*CREATEDOMAINCH_DOMCHAR*(20);
*CREATEDOMAIN_CH_DOMVARCHAR*(20);
CREATE DOMAIN TI_DOM TINYINT;
CREATE DOMAIN ID_DOM INTEGER;
CREATE DOMAIN RE_DOM REAL;
CREATE DOMAIN FL_DOM FLOAT;

```
CREATE DOMAIN DATE_DOM DATE;
CREATE TABLE doctor (

doc_no ID_DOM,
doc_name CH_DOM,
doc_address CH_DOM,
doc_basic FL_DOM,
specialization CH_DOM
primary key(doc_no)
);
CREATE TABLE patient (

p_no ID_DOM,
p_name CH_DOM,
p_address CH_DOM,
doc_no ID_DOM
primary key(p_no)
);
COMMIT

FINISH
```

## QUERY PROCESSING IN A LOOSELY-COUPLED SYSTEM :

**Query no : 1**

select emp_no emp_name address

where basic > 2000 or dept_no = 5

IT'S A LOCAL QUERY !!!

DIRECT THE QUERY :

select emp_no emp_name address
where basic > 2000 or dept_no = 5

TO **factory** DATABASE.

**Query No : 2**

select emp_no emp_name address

where basic > 2000 and specialization = opth and subject = history

NOW BREAK THE ORIGINAL QUERY AND DISTRIBUTE IT TO THE FOLLOWING DATABASES :

Distribute the following subquery :

select t_no t_name t_address
where t_basic > 2000 and subject = history

TO THE **school** DATABASE.

Distribute the following subquery :

---

select doc_no doc_name doc_address
where doc_basic > 2000 and specialization = opth

TO THE **hospital** DATABASE.

## QUERY PROCESSING IN A TIGHTLY-COUPLED SYSTEM :

**Query No : 1**

select emp_no emp_name address

where basic > 2000 or dept_no = 5

IT'S A LOCAL QUERY !!!

DIRECT THE QUERY :

select emp_no emp_name address
where basic > 2000 or dept_no = 5

TO **factory** DATABASE.

**Query No : 2**

select emp_no emp_name address

where specialization = opth and subject = biology and basic > 3500

NOW BREAK THE ORIGINAL QUERY AND DISTRIBUTE IT TO THE FOLLOWING DATABASES :

Distribute the following subquery :

select t_no
where subject = biology

TO THE **school** DATABASE.

Distribute the following subquery :

---

select doc_no
where specialization = opth

TO THE **hospital** DATABASE.

Distribute the following subquery :

---

select emp_no emp_name address
where basic > 3500

TO THE **factory** DATABASE.

**Query No : 3**

---

select subject specialization

where basic > 4000

Distribute the following subquery :

---

select subject t_no

TO THE **school** DATABASE.

Distribute the following subquery :

select specialization doc_no

TO THE **hospital** DATABASE.

Distribute the following subquery :

select emp_no
where basic > 4000

TO THE **factory** DATABASE.

# Chapter 6

# CONCLUSION

The purpose of this dissertation was to design an environment where a federated structure may grow gradully starting from a single database. Any database can also join or withdraw from federation causing a restructuring. This structure is more flexible than the conventional approach where the database to be used for the federation must exist before the federation is actually built.

The main emphasis was on the query processing in both loosely-coupled and tightly-coupled system. Two algorithms are suggested for this purpose which have been implemented in VAX/VMS environment. It was the plan of the author to include the temporal dimension in this DYNAFED, that would store alongwith information on entities and relationships, the entire history of the database. This could not be done, however for time constraint. This may be taken as a future continuation of this work.

# Chapter 7

# REFERENCES


1. D Heimberg and D McLeod , " A federated architechture for information management." ACM Trans Off Inf. Syst. Vol.3,No.3,pp 253-278,1985.

2. A P Seth and J A Larson , " Federated database systems for managing distributed, homogeneous and autonomous databases." ACM Computing Surveys. Vol.22,No.3,pp 183-236,1990.

3. M T Ozsu and P Valduriez," Principles of Distributed Database Systems", Prentice-Hall, 1991.

4. A Bagchi , " DYNAFED - A Dynamic Federation of Relational Databases in Data Management" : New Dimensions and Perspectives, International Journal Services, pp 113-128,1993.
   Calcutta,INDIA.

5. J A Larson,S.B.Navathe and R Elmasari," A theory of attribute equivalance in databases with application to schema integration." IEEE Trans. on Software Engg . Vol. 15,No. 4,pp 449-463,1989.

6. P. Barik , " Design of Dynamic Federation of Relational Databases ", Technical Report , ISI, Calcutta, 1994.