ALKALDI, WEJDAN ABDULLAH, M.S. Query Optimization in XML Based Information Integration for Queries Involving Aggregation and Group By. (2009) Directed by Dr. Fereidoon Sadri. 37 pp.

This thesis addresses optimization and processing of queries involving aggregation and group-by in the *semantic-model* approach to information integration. Query processing algorithms *materialization*, *subqueries*, and *wrapper* have been extended for such aggregate queries. Algorithms have been presented for two cases: In the first case information at different sources are disjoint, while in the second case information sources may contain overlapping information.

Keywords: Information integration; overlapping information; query optimization.

QUERY OPTIMIZATION IN XML BASED INFORMATION INTEGRATION FOR QUERIES INVOLVING AGGREGATION AND GROUP BY

by

Wejdan Abdullah Alkaldi

A Thesis Submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2009

Approved by

Committee Chair

To my loving parents.
To my father, Abdullah, who inspired me with his knowledge, wisdom, and guidance to be a better person.
To my mother, Sarah, whos encouragement and constant love sustains me throughout my life.

To my husband, Majed, thank you for believing in me; and supporting me to further my studies.

APPROVAL PAGE

This thesis has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair_____

Committee Members_____

_____

_____
Date of Acceptance by Committee

_____
Date of Final Oral Examination

ACKNOWLEDGMENTS

I would like to express my gratitude to all the people at UNCG Computer Science Department who gave me the possibility to complete this thesis. I am deeply indebted to my supervisor Prof. Fereidoon Sadri whose help, stimulating suggestions, and encouragement helped me in all the time of research for and writing of this thesis.

Especially, I would like to give my special thanks to my husband Majed whose patient love and support enabled me to complete this work.

TABLE OF CONTENTS

Page

CHAPTER

CHAPTER I

INTRODUCTION

Very large volumes of data are available in electronic form in many different systems and formats such as relational databases, XML data sources, spreadsheets, and on the web. The volume of data has been growing steadily over decades. This explosion of information comes with the need for advanced data management. Information integration is at the heart of needed functionalities for a large number of applications. This thesis presents an investigation into efficient query processing in information integration systems. More specifically, extensions to various query processing algorithms for the *semantic-model* approach to information integration has been presented to handle queries involving aggregation and group-by operations efficiently.

In recent years and decade, significant research has been conducted on information integration systems. The main goal of these systems is to combine information residing at different sources, and to provide the user with a unified view of these information. This thesis extends the work presented in [1, 2]. Three major algorithms, *materialization*, *subqueries*, and *wrapper* and a number of variations on these algorithms were introduced in these works. This thesis presents extensions to these algorithms for the optimization and processing of queries involving aggregation and group-by. The usual set of aggregation functions, *sum*, *count*, *average*, *min*, and *max*, have been covered. Different information sources, in general, can contain some overlapping data. Algorithms are presented for this general case, as

well as for the special case where data at the information sources are disjoint.

This thesis is organized as follows: Chapter 2 discusses the relevant background. Information integration is discussed in Section 2.1, and the Semantic-Model approach to information integration is reviewed in Section 2.2. Sections 2.3 and 2.4 are devoted to important definitions used throughout this thesis. Chapter 3 extends the *Materialization Approach* algorithm to handle queries involving aggregations and group-by. Chapter 4 shows how to extend the *Subqueries Approach* to handle queries involving aggregations and group-by, with emphasis on query processing efficiency. Aggregate operations *min* and *max* are covered in Section 4.1, *count* and *sum* in Section 4.2, and *avg* in Section 4.3. Algorithms are presented for both overlapping and non-overlapping sources. In Chapter 5, we present the third approach, the *Wrapper*. We present theoretical results regarding when the *chase* should (and should not) be applied in this approach. These results provide further optimization by avoiding the chase task when it is not needed. Finally, in Chapter 6, we provide a comprehensive example that demonstrate all of our algorithms produce the same answer to a user query. This is an indication of the correctness of our algorithms. Conclusions are presented in Chapter 7.

CHAPTER II

BACKGROUND

2.1 Information Integration

Large amounts of data are available in electronic form on the web and elsewhere and the volume is growing rapidly. However, the Web's browsing paradigm does not support many information management tasks as mentioned in [3]. Combining information residing at different sources, and providing the user with a unified view of the information is the problem of designing information integration systems. This problem has become extremely important in current real world applications, and is, in addition to significant practical importance, characterized by a number of issues that are interesting from a theoretical point of view [5].

Most information integration systems are characterized by an architecture based on a global schema and a set of sources. The sources contain the real data, while the global schema provides a reconciled, integrated, and virtual view of the underlying sources. Modeling the relation between the sources and the global schema is therefore a crucial aspect [5]. For this reason, two general approaches have been introduced:

(i) global-as-view (GAV), requires that the global schema is expressed in terms of the data sources, and

(ii) local-as-view (LAV), requires the global schema to be specified independently. The connections between the global schema and the sources are founded by defining every source as a view over the global schema.

In the past few years, many researches have investigated issues in information integration systems. Here, we will consider the four techniques presented in [2], namely, materialization, subqueries, optimized subqueries, and wrapper approaches to efficient query processing. We extend these techniques to handle queries with aggregation and group-by. Our emphasis is on efficient processing of these queries.

2.2 The Semantic-Model Approach to Information Integration

The *semantic-model* (SM) approach to information integration and interoperability was first introduced in [4]. SM system have two types of sites, information sources and coordinators (also called *mediators*). A coordinator oversees query decomposition and execution. User queries can be submitted at any site [1].

The Semantic-Model approach enables interaction between (i) data sources that store data using XML or relational data models, and (ii) relational mediators whose schema's conform to a Semantic-Model. The information at each source is viewed as a collection of (logical) binary relations, which is called the semantic-model view [2].

In this paper, we are working on sources that have XML or relational data in their local schemas. However, in all our examples, we are showing the data in the form of the semantic-model relations for simplicity.

In this thesis, we extend the query processing algorithms introduced in [1, 2] to handle queries that involve aggregation and group-by operations. We will provide a short description of these algorithms below.

The Materialization Approach

This is the base query-processing approach against which we evaluate other

approaches. In the simple materialization approach, we materialize the SM view relations that appear in the user query, and execute the query on the materialized relations [1]. This approach is not efficient in general, and it is used mainly for comparison.

The Subqueries Approach

The subqueries approach is based on generating and executing local and inter-source subqueries for the user query, and then merging these partial results to obtain the query answer. A local subquery is executed on data from a single source and can be executed locally at that source. An inter-source subquery is executed on data from multiple sources. In this case some data transmission will be required [2].

The Optimized-Subquery Approach

In the optimized-subquery approach we use semantic constraints to eliminate, to the extent possible, inter-source processing. Based on key and foreign-key constraints that are relevant to the query, all or some of inter-source subqueries may be redundant and need not be evaluated [2].

The Wrapper Approach

The wrapper approach, generates only one local subquery per data source. This subquery extracts from the source the minimum amount of information that is needed to answer the user query. It is called "wrapper" approach because this extraction can be viewed as a (query-specific) wrapper that collects the needed information from each source. Compared to local subqueries, the information extracted from each source in the wrapper approach is richer than the result of the local subquery on the same source, thus making it possible to obtain the full answer

to the user query by further processing. Intuitively, the information collected by the local (wrapper) query corresponds to the full outer-join of the relations involved. In a large class of applications, an efficient chase-based algorithm [6] can be applied to the extracted information to obtain the full answer to the user query [1, 2].

2.3 Overlapping Sources

In this thesis, we will extend query processing algorithms for two possible cases:
Case 1: Information sources do not have any overlapping information, and
Case 2: Information sources have overlapping information (see definition below).

Definition 1 (Overlapping) *Two sources s_1 and s_2 are said to have* overlapping data *(or to be* overlapping*) when, for a predicate $p(A, B)$ in the semantic model where A is the key, s_1 has a tuple (a_1, b_1) in p and s_2 has a tuple (a_2, b_2) in p where $a_1 = a_2$.*

Note that b_1, b_2, or both can be NULL. If they are both values but $b_1 \neq b_2$, then the two sources have *inconsistent data* (are *inconsistent*). That means the combination of their data violates a constraint.

Example 1 *Assume we have two sources s_1 and s_2, and the semantic model has a predicate* `id-city`*, with the following values. Assume id is the key for* `id-city`*.*

id	*city*
100	–
120	*Greensboro*

id	*city*
100	*Charlotte*
110	*Raleigh*

These sources have overlapping information, but no inconsistencies. ■

2.4 Aggregation Queries

The general form of a query that involves aggregation and group-by is

```
select   W, Agg(M)
from     relations
where    conditions P
group by N
```

Where M is an attribute, N is a set of attributes, $W \subseteq N$, Agg is an aggregates operation such as *min, max, count, sum,* and *average.* Each condition in the set of conditions P has one of the following forms: `A op B`, `A op K`, and `K op A`, where `A` and `B` are attributes, `K` is a constant, and `op` is an arithmetic comparison like $=$, $\neq$, $<$, $>$, $\leq$, and $\geq$.

We will refer to this form of query as *Aggregation Query Form* in the next chapters.

CHAPTER III

MATERIALIZATION APPROACH

In this chapter we extend the *Materialization Approach* to handle queries involving aggregations and group-by. All queries are processed as described bellow for overlapping as well as non-overlapping sources.

3.1 Processing Queries with Materialization Approach

In this approach, overlapping sources and non-overlapping sources will be handled similarly. Suppose we have a query in *Aggregation Query Form* shown below for convenience.

```
select   W, Agg(M)
from     relations
where    conditions P
group by N
```

Then to execute the query using the Materialization Approach, we follow the following steps:

1. At each source materialize the relations that appear in the from clause. We can enforce those conditions in P that have the form $A\ op\ K$ or $K\ op\ A$ where A is an attribute and K is a constant at this time.

2. Send the results from each source to the mediator.

3. In the mediator, combine (union) binary relations for each selection in the from clause of the query. Inconsistencies, if any, are detected at this step.

4. execute a modified query at the mediator. The modified query is obtained by removing those conditions that were enforced in step 1 from the original query.

We will illustrate these steps with the following example.

Example 2 *Suppose we have two sources s_1 and s_2 in some federated universities databases containing the following data. Note that, sources can have XML or relational data in their local schemas. Here we are showing the data in the form of the semantic-model relations for simplicity.*

source s_1

id	*major*
100	*English*
110	*Math*

id	*gpa*
100	*3.2*
110	*3.0*

id	*tuition*
100	*in-state*
110	*out-of-state*

source s_2

id	*major*
120	*English*

id	*gpa*
120	*2.8*

id	*tuition*
120	*in-state*

Now consider the following query that lists the minimum GPA of in-state students enrolled in each major:

```
select   id-major.major, min(id-gpa.gpa)
from     id-gpa, id-major, id-tuition
where    id-gpa.id = id-major.id and
         id-gpa.id = id-tuition.id and
         id-tuition.tuition = 'in-state'
group by id-major.major
```

Using the Materialized approach to execute this query, we do the following:

1. For each source, we enforce the condition `id-tuition.tuition = 'in-state'` *in the where-clause to obtain*

source s_1

id	*major*
100	*English*
110	*Math*

id	*gpa*
100	*3.2*
110	*3.0*

id	*tuition*
100	*in-state*

source s_2

id	*major*
120	*English*

id	*gpa*
120	*2.8*

id	*tuition*
120	*in-state*

2. These tables are sent to the mediator.

3. Binary relations for each relation in the query are merged to obtain:

id	*major*
100	*English*
110	*Math*
120	*English*

id	*gpa*
100	*3.2*
110	*3.0*
120	*2.8*

id	*tuition*
100	*in-state*
120	*in-state*

4. The following query, which is the modified version of the original query with the enforced condition removed, is executed on these relations in the mediator

```
select   id-major.major, min(id-gpa.gpa)
from     id-gpa, id-major, id-tuition
where    id-gpa.id = id-major.id and
         id-gpa.id = id-tuition.id
group by id-major.major
```

The final result is:

major	*min(gpa)*
English	*2.8*

CHAPTER IV

SUBQUERIES APPROACH

In this chapter we will show how to extend the *Subqueries Approach* to handle queries involving aggregations and group by efficiently. We will also discuss the issue of inter-source subqueries and investigate different strategies for their execution.

We will see that for some aggregations we need to handle overlapping sources and non-overlapping sources differently and for others we can handle them similarly.

The subqueries algorithm, in general, consists of three phases:

1. **Local and inter-source subqueries phase**. In this phase local and inter-source subqueries are generated and submitted for execution to the sources. For simple non-aggregate queries, the subqueries are translations of user query to the sources' local data format (for example, XML). Translation for aggregate queries is more involved and will be discussed further below. Results of the subqueries are sent to the mediator.

2. **Collection and merge phase**. In this phase, the mediator "combines" the results of local and inter-source subqueries. For simple (non-aggregate) queries, this is often the end of the processing and the result is the final answer to user query. For queries involving aggregation and group-by, additional processing is needed.

3. **Additional processing phase**. In this phase additional processing, in the form of queries executed on the result of previous phase, is carried out by the

mediator to obtain the final answer to user query.

For each aggregation type, namely, min, max, count, sum, and average, we need to develop the algorithms for the query translation (in Phase 1) and additional processing (in Phase 3). These will be discussed below.

4.1 Min and Max Aggregations

In min and max aggregations, duplicate tuples do not affect the results. So, we will treat overlapping sources and non-overlapping sources similarly. Suppose we have a query in *Aggregation Query Form* with Min/Max aggregation. Then the execution of this query using the subqueries algorithm, consists of three phases:

1. **Local and inter-source subqueries phase**. For min and max aggregation, translated subqueries perform the user query, including group-by and aggregation, on the local data. Then the results are sent to the mediator.

2. **Collection and merge phase**, which is done in the mediator, simply unions (without duplicate removal) the results of the previous phase.

3. **Additional processing phase**, which is done in the mediator too. In this phase we apply the group-by and aggregation on the collected results.

The following example will show how the three phases will work.

Example 3 *Suppose we have two sources s_1 and s_2 in some federated universities databases containing the following data. We are assuming that a student can take courses from any university in the federation, but some student information, such as GPA, is only available at the student's "home" university. Note that, in general, sources can have XML or relational data in their local schemes. Here we are showing the data in the form of the semantic-model relations for simplicity.*

source s_1

id	*course*
100	*Eng111*
110	*Eng111*
120	*Eng111*

id	*gpa*
100	*3.2*
110	*3.0*

source s_2

id	*course*
120	*Art333*

id	*gpa*
120	*2.8*

Now consider the following query that lists the minimum GPA of students registered in each course:

```
select   id-course.course, min(id-gpa.gpa)
from     id-gpa, id-course
where    id-gpa.id = id-course.id
group by id-course.course
```

Phase 1: The results of local and inter-source subqueries are shown below. The first two are results of local subqueries at the two sources, the third is the result of an inter-source subquery. These results will be sent to the mediator.

course	*min(gpa)*
Eng111	*3.0*

course	*min(gpa)*
Art333	*2.8*

course	*min(gpa)*
Eng111	*2.8*

Phase 2: The results of Phase 1 are combined at the mediator:

course	*min(gpa)*
Eng111	*3.0*
Art333	*2.8*
Eng111	*2.8*

Phase 3: A final query is executed by the mediator on the result of Phase 2 to obtain the answer to user query:

course	*min(gpa)*
Eng111	*2.8*
Art333	*2.8*

4.2 Count and Sum Aggregations

In `Count` and `Sum` Aggregations, duplicate tuples do affect the results. So, we will treat overlapping sources and non-overlapping sources separately. The same algorithm that will apply on `count` will apply on `sum` as well.

I. Non-overlapping sources

In a non-overlapping sources, to execute a query that involves aggregation `Count` or `Sum` we need three phases :

1. **Local and inter-source subqueries phase**. For a query that involves count or sum aggregation, translated subqueries perform the user query (including group-by and aggregation) on the local data. Then the results are sent to the mediator.

2. **Collection and merge phase**, which is done in the mediator, simply unions (without duplicate removal) the results of the previous phase.

3. **Additional processing phase**, which is done in the mediator too. In this phase we apply the group-by and the `sum` aggregation on the collected results.

The following example will show how the three phases will work.

Example 4 *Suppose we have two non-overlapping sources s_1 and s_2 as shown in Example 3. Now consider the following query that lists the number of students registered in each course with $GPA \leq 3.0$:*

```
select   id-course.course, Count(id-gpa.id)
from     id-gpa, id-course
where    id-gpa.id = id-course.id and
         id-gpa.gpa <= 3.0
group by id-course.course
```

Phase 1: The results of local and inter-source subqueries are shown below. The first two are results of local subqueries at the two sources, the third is the result of an inter-source subquery. These results will be sent to the mediator.

course	*count(id)*
Eng111	*1*

course	*count(id)*
Art333	*1*

course	*count(id)*
Eng111	*1*

Phase 2: The results of Phase 1 are combined at the mediator:

course	*count(id)*
Eng111	*1*
Art333	*1*
Eng111	*1*

Phase 3: A final query (with Sum aggregation) is executed by the mediator on the result of Phase 2 to obtain the answer to the user query:

course	*count(id)*
Eng111	*2*
Art333	*1*

II. Overlapping sources

For overlapping sources, to execute a query that involves aggregation `Count` or `Sum`, the three phases are somewhat different since duplicate tuples affect the results :

1. **Local and inter-source subqueries phase**. For a query that involves count or sum aggregation, translated subqueries perform the user query (without group-by nor aggregation) on the local data. Then the results are sent to the mediator.

2. **Collection and merge phase**, which is done in the mediator, simply unions (with duplicate removal) the results of the previous phase.

3. **Additional processing phase**, which is done in the mediator too. In this phase we apply the user query (with group-by and aggregation) on the collected results.

The following example will show how the three phases will work.

Example 5 *Suppose we have two overlapping sources s_1 and s_2 in some federated universities databases containing the following data. We are assuming that a student can take courses from any university in the federation, and some student information, such as City, is available at other universities databases. Note that, in general, sources can have XML or relational data in their local schemes. Here we are showing the data in the form of the semantic-model relations for simplicity.*

source s_1

id	*course*
100	*Eng111*
110	*Eng111*
120	*Eng111*

id	*city*
100	*Greensboro*
110	*Charlotte*
120	*Greensboro*

source s_2

id	*course*
120	*Art333*

id	*city*
120	*Greensboro*

Now consider the following query that lists the number of students from Greensboro registered in each course:

```
select   id-course.course, count(id-city.id)
from     id-city, id-course
where    id-city.id = id-course.id and
         id-city.city = 'Greensboro'
group by id-course.course
```

Phase 1: The results of local and inter-source subqueries are shown below. The first two are results of local subqueries at the two sources, the third and the fourth are the result of inter-source subqueries. These results will be sent to the mediator.

id	*course*	*city*
100	*Eng111*	*Greensboro*
110	*Eng111*	*Charlotte*
120	*Eng111*	*Greensboro*

id	*course*	*city*
120	*Art333*	*Greensboro*

id	*course*	*city*
120	*Art333*	*Greensboro*

id	*course*	*city*
120	*Eng111*	*Greensboro*

Phase 2: The results of Phase 1 are combined at the mediator with duplicate tuples removal:

id	*course*	*city*
100	*Eng111*	*Greensboro*
110	*Eng111*	*Charlotte*
120	*Eng111*	*Greensboro*
120	*Art333*	*Greensboro*

Phase 3: A final query is executed by the mediator on the result of Phase 2 to obtain the answer to user query:

course	*count(id)*
Eng111	*2*
Art333	*1*

4.3 Average Aggregation

In `Avg` Aggregation, duplicate tuples do affect the results. So, we will treat overlapping sources and non-overlapping sources separately.

I. Non-overlapping sources

In non-overlapping sources, to execute a query that involves aggregation `Avg`, we can use $average = \frac{sum}{count}$ to design our algorithm. Conceptually, we evaluate the following query using the algorithm discussed in previous section.

```
select   W, sum(M), count(M)
from     relations
where    conditions P
group by N
```

Then, as the final processing, the average is calculated as $\frac{sum}{count}$ for each group. The following example will show how these phases work.

Example 6 *Suppose we have two non-overlapping sources s_1 and s_2 as shown in Example 3. Now consider the following query that lists the average gpa of students registered in each course:*

```
select   id-course.course, Avg(id-gpa.gpa)
from     id-gpa, id-course
where    id-gpa.id = id-course.id
group by id-course.course
```

Phase 1: The results of local and inter-source subqueries are shown below. The first two are results of local subqueries at the two sources, the third is the result of an inter-source subquery. These results will be sent to the mediator.

course	*sum(gpa)*	*count(gpa)*
Eng111	*6.2*	*2*

course	*sum(gpa)*	*count(gpa)*
Art333	*2.8*	*1*

course	*sum(gpa)*	*count(gpa)*
Eng111	*2.8*	*1*

Phase 2: The results of Phase 1 are combined at the mediator:

course	*sum(gpa)*	*count(gpa)*
Eng111	*6.2*	*2*
Art333	*2.8*	*1*
Eng111	*2.8*	*1*

After that we sum the numbers of `sum(gpa)` *and* `count(gpa)` *for each group and we get*

course	*sum(gpa)*	*count(gpa)*
Eng111	*9.0*	*3*
Art333	*2.8*	*1*

Phase 3: execute the user query with average aggregation on the result of Phase 2 using the fact average = sum/count to obtain the answer:

course	*avg(gpa)*
Eng111	*3.0*
Art333	*2.8*

II.Overlapping sources

The same approach discussed in previous section, can be used here too. Of course, the overlapping sources algorithm should be used for *sum* and *count*. Alternatively, the overlapping source algorithm can be applied to directly compute the average. The modified algorithm (for average) is straight forward and we omit the details here.

4.4 Reducing Inter-Source Subqueries

To execute any user query using subqueries approach, we need to preform n^k subqueries where n is the number of sources and k is the number of the relations in the query. Only n subqueries are done locally and the rest are inter-source subqueries. However, in some cases, all or some of these inter-source subqueries are redundant and will not be evaluated. We can reduce the number of inter-source subqueries (or eliminate them in best cases) if we find a key and a foreign-key constraints that are relevant to the user query. [2]

The following Theorem from [1] will illustrate that.

Theorem 1 *[1] Given a user query involving the natural join of two or more relations $r^1, ..., r^k$, if the local-join graph restricted to the query relations $r^1, ..., r^k$ contains a directed spanning tree, then no inter-source processing is needed for this query.*

The following example will demonstrate this theorem.

Example 7 *Suppose we have a query in* Aggregation Query Form *that uses tables $u(A, B), v(C, A),$ and $w(D, C)$. Suppose that attribute A is the key for u and that*

a foreign-key constraint holds from $v.A$ to $u.A$. Also, suppose that attribute C is the key for v and that a foreign-key constraint holds from $w.C$ to $v.C$. Then, the local-join graph for these relations is as follows:

$$u \longleftarrow v \longleftarrow w$$

The above graph, has a directed spanning tree that goes through w, v then u. In this case, we can eliminate all inter-source subqueries, and only local subqueries need to be executed.

Suppose we have a query in *Aggregation Query Form*. If the condition of Theorem 1 does not hold then some inter-source processing is needed. [1]

4.5 Execution of Inter-source Subqueries

Local subqueries are executed locally within their sources. However, inter-source subqueries use data from more than one source, and could be executed at one of the sources or at the mediator. If all the sources have relatively small amount of data, then the execution of inter-source subqueries will be done in the mediator. In the other hand, if at least one of the sources have a large amount of data, then it is better to execute the inter-source subquery at this particular source. By doing that, we avoid transforming large amounts of data.

We will discuss the case when all sources have a small amount of data. In this case, the inter-source subqueries will be executed in the mediator.

A. Min and Max Aggregations

Min and Max are handled similarly. In min and max aggregations, duplicate tuples do not affect the results. So, we will treat overlapping sources and non-overlapping sources similarly. The algorithm consists of five phases:

1. **subqueries phase**. We translate the user query into subqueries (local and inter-source subqueries) that perform as the user query, including group-by and aggregation min/max. Then we send the inter-source subqueries to the mediator and the local subqueries to the local source they use.

2. **local subqueries phase**, which is done locally in sources. We execute the local subqueries and send the results to the mediator.

3. **inter-source subqueries phase**, which is done in the mediator. We execute the inter-source subqueries and keep the results in the mediator.

4. **Collection and merge phase**. This phase is done in the mediator, simply unions (without duplicate removal) the results of the previous phases.

5. **Additional processing phase**, which is done in the mediator too. In this phase we apply the user query on the collected results with respect to the group-by to get the final results.

The following example will show how these phases will work.

Example 8 *Suppose we have two sources s_1 and s_2 with tables* `id-source` *and* `id-gpa` *as in Example 3*

source s_1

t

id	*course*
100	*Eng111*
110	*Eng111*
120	*Eng111*

u

id	*gpa*
100	*3.2*
110	*3.0*

source s_2

t

id	*course*
120	*Art333*

u

id	*gpa*
120	*2.8*

Now consider the following query that lists the maximum GPA of students with gpa ≤ 3.0 registered in each course:

```
select   id-course.course, max(id-gpa.gpa)
from     id-gpa, id-course
where    id-gpa.id = id-course.id
         id-gpa.gpa <= 3.0
group by id-course.course
```

To execute this query using the Simi-Subqueries approach, we need five phases:

1. **subqueries Phase:** *In this phase we will send the inter-source subqueries (that will use (t_1, s_2) relations and (t_2, s_1) relations) to the mediator and send the local subqueries (that will use (t_1, s_1) and (t_2, s_2)) to source s_1 and s_2.*

2. **Local subqueries phase:** *Here, we run in source s_1 the local subquery that uses the relations (t_1, s_1); and the same in source s_2 we run the subquery with (t_2, s_2). Then we send the results in s_1 and s_2 to the mediator. That will result in sending the following data to the mediator.*

id	*course*	*gpa*
110	*Eng111*	*3.0*

id	*course*	*gpa*
120	*Art333*	*2.8*

3. **inter-source subqueries phase:** *We run the inter-source subqueries that uses (t_1, s_2) relations and (t_2, s_1) relations. The first subquery will give the following table and second one will result in nothing*

id	*course*	*gpa*
120	*Eng111*	*2.8*

4. **Collection and merge phase:** *In the mediator, we union the previous results without duplicate removal.*

5. **Additional processing phase:** *In this phase, we apply the user query on the previous results and we get*

course	*max(gpa)*
Eng111	*3.0*
Art333	*2.8*

B. Count and Sum Aggregations

Count and Sum are handled similarly. In Count and Sum aggregations, duplicate tuples affect the results. So, we will treat overlapping sources and non-overlapping sources separately.

I. Non-overlapping Sources: Using a non-overlapping sources, we need five phases (similar to the ones introduced in the previous section) in order to execute a query with count or sum aggregation.

1. **subqueries phase**. We translate the user query into subqueries(local and inter-source subqueries) that perform as the user query with group-by and count/ sum aggregation. Then we send the inter-source subqueries to the mediator and the local subqueries to the local source they use.

2. **local subqueries phase**, which is done locally in sources. We run the local subqueries (that needs only the local source) and we send the results to the mediator.

3. **inter-source subqueries phase**, which is done in the mediator. We run the inter-source subqueries (that uses more than one source) and we keep the results in the mediator.

4. **Collection and merge phase**. This phase is done in the mediator, simply unions (without duplicate removal) the results of the previous phases.

5. **Additional processing phase**, which is done in the mediator too. In this phase we just sum the collected results whether we had *sum* or *count* aggregation in the user query, with respect to the group-by to get the final results.

II. Overlapping Sources: In this type of sources, we use the same phases introduced before but with different description. So, the new phases will be

1. **subqueries phase**. We translate the user query into subqueries (local and inter-source subqueries) that perform a modified query which is the same as the user query except (1) *select clause* has been replaced by *select **, and (2)group-by clause has been removed. Then we send the inter-source subqueries to the mediator and the local subqueries to the local source they use.

2. **local subqueries phase**, which is done locally in sources. We run the local subqueries (that needs only the local source) and we send the results to the mediator.

3. **inter-source subqueries phase**, which is done in the mediator. We run the inter-source subqueries (that uses more than one source) and we keep the results in the mediator.

4. **Collection and merge phase**. This phase is done in the mediator, simply unions (with duplicate removal) the results of the previous phases.

5. **Additional processing phase**, which is done in the mediator too. In this phase we apply the user query on the collected results with respect to the group-by and the aggregation to get the final results.

This example will show how these phases will work.

Example 9 *Suppose we have two sources* s_1 *and* s_2 *with tables* `id-course` *and* `id-gpa` *as below*

source s_1

t

id	*course*
100	*Eng111*
110	*Eng111*
120	*Eng111*

u

id	*gpa*
100	*3.2*
110	*3.0*
120	*2.8*

source s_2

t

id	*course*
120	*Art333*

u

id	*gpa*
120	*2.8*

Now consider the following query that lists the number of students with gpa ≤ 3.0 *registered in each course:*

```
select   id-course.course, count(id-gpa.gpa)
from     id-gpa, id-course
where    id-gpa.id = id-course.id
         id-gpa.gpa <= 3.0
group by id-course.course
```

To execute this query using the above algorithm:

1. **subqueries Phase**. *In this phase we will send the inter-source subqueries (that will use* (t_1, s_2) *relations and* (t_2, s_1) *relations) to the mediator and send the local subqueries (that will use* (t_1, s_1) *and* (t_2, s_2)*) to source* s_1 *and* s_2.

2. **Local subqueries phase**. *Here, we run in source* s_1 *the local subquery (with select all) that uses the relations* (t_1, s_1)*, and the same in source* s_2 *we run the subquery with* (t_2, s_2) *relations. Then we send the results in* s_1 *and* s_2 *to the mediator. That will result in sending the following data to the mediator:*

id	*course*	*gpa*
110	*Eng111*	*3.0*
120	*Art333*	*2.8*

id	*course*	*gpa*
120	*Art333*	*2.8*

3. **inter-source subqueries phase.** *Here we run the inter-source subqueries that uses* $((t_1, s_2)$ *relations and* $((t_2, s_1)$ *relations. That will give the following tables:*

id	*course*	*gpa*
120	*Eng111*	*2.8*

id	*course*	*gpa*
120	*Art333*	*2.8*

4. **Collection and merge phase**. *In the mediator, we union the previous results with duplicate removal. That will result in:*

id	*course*	*gpa*
110	*Eng111*	*3.0*
120	*Eng111*	*2.8*
120	*Art333*	*2.8*

5. **Additional processing phase**. *In this phase, we apply the user query and we get:*

course	*count(gpa)*
Eng111	*2*
Art333	*1*

C. Average Aggregation

Here we will treat overlapping sources and non-overlapping sources separately since duplicate tuples do affect the results.

I. Non-overlapping Sources: Using a non-overlapping sources, we need five phases in order to execute a query with avg aggregation.

1. **subqueries phase**. We translate the user query into subqueries (local and inter-source subqueries) that perform as the user query with sum and count instead of average(since $average = sum/count$) with respect to the group by. Then we send the inter-source subqueries to the mediator and the local subqueries to the local source they use.

2. **local subqueries phase**, which is done locally in sources. We run the local subqueries (that needs only the local source) and we send the results to the mediator.

3. **inter-source subqueries phase**, which is done in the mediator. We run the inter-source subqueries (that uses more than one source) and we keep the results in the mediator.

4. **Collection and merge phase**. This phase is done in the mediator, simply sum (with respect to the group-by) the results of the previous phases.

5. **Additional processing phase**, which is done in the mediator too. In this phase we just calculate the average using the sum and count numbers we found in the collected results with respect to the group-by to get the final results.

II. Overlapping Sources: Here, we will use the same phases title introduced before but again with different description. So, phases new description will be:

1. **subqueries phase**. We translate the user query into subqueries (local and inter-source subqueries) that perform the modified query as explained previ-

ously. Then we send the inter-source subqueries to the mediator and the local subqueries to the local source they use.

2. **local subqueries phase**, which is done locally in sources. We run the local subqueries (that needs only the local source) and we send the results to the mediator.

3. **inter-source subqueries phase**, which is done in the mediator. We run the inter-source subqueries (that uses more than one source) and we keep the results in the mediator.

4. **Collection and merge phase**. This phase is done in the mediator,simply unions (with duplicate removal) the results of the previous phases.

5. **Additional processing phase**, which is done in the mediator too. In this phase we apply the user query, including the group-by and aggregation average, on the collected results to get the final answer.

4.5.1 Performing early selections to improve efficiency of inter-source subquery execution

We can use the *where clause conditions* to minimize the amount of data have to be sent from each source to the mediator. That can be done by enforcing the P *condition* of type A *op* K or K *op* A at each source to filter out the tuples that will not be used. Then we send the results to the mediator where we enforce the rest of the conditions that we did not enforce locally at each source.

CHAPTER V

WRAPPER APPROACH

In the *Wrapper Approach*, we obtain the full answer using the outer-join and, in some cases, the *chase-based* algorithm introduced in [6].

5.1 Min and Max Aggregations

The same argument that will apply on `Min` will apply on `Max`. So, we will work with one of them. Consider a query in the *Aggregation Query Form* with `Min` aggregation, shown below for convenience.

```
select   W, Agg(M)
from     relations
where    conditions P
group by N
```

Theorem 2 *Let* U *be the set of attributes that appear in conditions* P *that have the form* A op K *or* K op A. *For any query in* Aggregation Query Form *with Min/Max aggregation and any functional dependency* $X \rightarrow Y$, *where* X *and* Y *are sets of attributes, if* W, U, *and* M *are disjoint from* Y, *then there is no need to do the chase with respect to* $X \rightarrow Y$. *Otherwise, chase should be applied.*

Proof. In this case values in attributes W, U, and M, will not change by chase. Hence, the group by and aggregation attribute values are not affected, and further, a tuple satisfies P after the chase if and only if it satisfies it before the chase. Keeping in mind that in the wrapper approach, chase is applied to the union of outer-join of

tables in each source, we notice the only possible change due to chase is changing the multiplicities of the groupings (by the W group by attributes). But min (and max) operations are not sensitive to multiplicities. Hence, the answer to the query will be the same if chase is not carried out, and hence, there is no need to do the chase. ■

The following examples demonstrate the cases where chase is needed and not needed.

Example 10 *To show a case where applying the chase is required, consider the information sources s_1 and s_2 containing the following data:*

source s_1

id	*city*
100	-
120	*Greensboro*

id	*name*
100	*Ann*
120	*Chi*

id	*year*
100	*2005*
120	*2009*

id	*gpa*
100	*2.8*
120	*2.9*

source s_2

id	*city*
100	*Greensboro*
110	*Raleigh*

id	*name*
100	-
110	*Bob*

id	*year*
100	-
110	*2008*

id	*gpa*
100	-
110	*3.2*

Assume the following functional dependencies hold: `id`→`city`, `id`→`name`, `id`→`year`, *and* `id`→`gpa`. *Now consider the query:*

```
select   id-city.city, id-name.name, Min(id-gpa.gpa)
from     id-city, id-name, id-year, id-gpa
where    id-city.id = id-name.id and
         id-name.id = id-year.id and
         id-year.id = id-gpa.id and
         id-year.year>2000
group by id-city.city, id-name.name
```

In the wrapper approach we get the following relation at the coordinator:

id	*city*	*name*	*year*	*gpa*
100	-	*Ann*	*2005*	*2.8*
120	*Greensboro*	*Chi*	*2009*	*2.9*
100	*Greensboro*	-	-	-
110	*Raleigh*	*Bob*	*2008*	*3.2*

If we do not apply the chase on any of the functional dependencies, then we will get the following (wrong) answer to the query:

city	*name*	*Min(gpa)*
Greensboro	*Chi*	*2.9*
Raleigh	*Bob*	*3.2*

Note that all FD's need to be applied according to Theorem 2. Now if we apply the chase on the functional dependencies, we obtain the following selection at the coordinator:

id	*city*	*name*	*year*	*gpa*
100	*Greensboro*	*Ann*	*2005*	*2.8*
120	*Greensboro*	*Chi*	*2009*	*2.9*
110	*Raleigh*	*Bob*	*2008*	*3.2*

Then the (correct) answer will be

city	*name*	*Min(gpa)*
Greensboro	*Ann*	*2.8*
Raleigh	*Bob*	*3.2*

Case 4: when $\mathtt{W} \cap \mathtt{Y} \neq \phi$ *and* $\mathtt{N} \cap \mathtt{Y} \neq \phi$*, chase* $\mathtt{X} \rightarrow \mathtt{Y}$*. In this example, we chase* $\mathtt{id} \rightarrow \mathtt{city}$ *and we get the same result as above.*

Example 11 *To show that we do not need to apply the chase on $X \rightarrow Y$ when W, M, and U are disjoint from Y, suppose we have s_1 and s_2 as shown below, and the functional dependency id → position*

source s_1

id	*position*
100	-
120	*grader*

position	*salary*
grader	*20$ /hour*

source s_2

id	*position*
100	*researcher*
110	*lab monitor*

position	*salary*
researcher	*23$/hour*
lab monitor	*18$/hour*

Now, consider the query:

```
select   id-position.id, Min(position-salary.salary)
from     id-position, position-salary
where    id-position.position = position-salary.position and
         position-salary.salary > 10
group by id-position.id
```

In the wrapper approach we get the following relation at the coordinator:

id	*position*	*salary*
120	*grader*	*20*
100	*researcher*	*23*
110	*lab monitor*	*18*

Since, in the query, position is only used for the join, then whether we apply the chase on id → position or not we will get the same result

id	*salary*
120	*20*
100	*23*
110	*18*

So, there is no need to chase id → position.

5.2 Count and Sum Aggregation

The same argument that will apply on `Count` will apply on `Sum`. So, we will work with one of them. In this type of aggregation, we will distinguish between overlapping sources and non-overlapping sources. Consider a query in *Aggregation's Query Form* with count aggregation.

Theorem 3 *For sources that have no overlapping information, for any query in* Aggregation Query Form *with Count/Sum aggregation and any functional dependency $X \rightarrow Y$, where X, $Y \subset R$, if W, U, and M are disjoint from Y, then there is no need to chase $X \rightarrow Y$. Otherwise, chase should be applied.*

Note: Theorem 3 does not hold for sources that have overlapping information. In the overlapping information database, we need to apply the chase on all functional dependencies.

The following examples demonstrate the cases where chase is needed and not needed in the non-overlapping and overlapping sources.

I. Non-Overlapping sources

This example shows that we do not need to chase $X \rightarrow Y$ when W, M, and U are disjoint from Y.

Example 12 *Suppose we have two sources s_1 and s_2 as shown below, and the functional dependency id → position*

source s_1

id	*position*
100	-
120	*grader*

position	*salary*
grader	*20*

source s_2

id	*position*
130	*researcher*
110	*lab monitor*

position	*salary*
researcher	*23*
lab monitor	*18*

Now, consider the query:

```
select   position-salary.salary, Count(id-position.id)
from     id-position, position-salary
where    id-position.position = position-salary.position and
         position-salary.salary > 10
group by position-salary.salary
```

In the wrapper approach we get the following relation at the coordinator:

id	*position*	*salary*
120	*grader*	*20*
130	*researcher*	*23*
110	*lab monitor*	*18*

Since, in the query, position is only used for the join, then whether we apply the chase on id → position or not we will get the same result

salary	*count(id)*
20	*1*
23	*1*
18	*1*

So, there is no need to chase id → position.

II. Overlapping sources

The following example shows that in overlapping sources, chase is needed in the wrapper approach for the `count` (and `sum`) aggregation.

Example 13 *Consider the information sources s_1 and s_2 of Example 10 with the following query:*

```
select   id-name.id, id-name.name, count (*)
from     id-name, id-city
where    id-name.id = id-city.id
group by id-name.id, id-name.name
```

In the wrapper approach we get the following relation at the coordinator:

id	*name*	*city*
100	*Ann*	-
100	*Ann*	*Greensboro*
110	*Bob*	*Raleigh*
120	*Chi*	*Greensboro*

It is easy to see that we need to apply the chase and eliminate duplicates before executing the group by and aggregation in order to get the right answer. ■

5.3 Average Aggregation

The same result holds for queries with average aggregation. Consider a query in *Aggregation Query Form* with average aggregation.

Theorem 4 *For sources that have no overlapping information, for any query in* Aggregation Query Form *with average aggregation and any functional dependency $X \rightarrow Y$, where X, $Y \subset R$, if W, U, and M are disjoint from Y, then there is no need to chase $X \rightarrow Y$. Otherwise, chase should be applied.*

Note: Theorem 4 does not hold for sources that have overlapping information. In the overlapping information database, we need to apply the chase on all

functional dependencies.

CHAPTER VI

COMPREHENSIVE COMPARISON OF ALGORITHMS

Running a query using any of the previous approaches should give the same results. In this chapter, we will show that the answer for a query using the Materialization approach, will be the same using Subqueries approach or Wrapper approach. in following example we are using an overlapping sources.

Example 14 *Suppose we have two XML sources* s_1 *and* s_2 *with different formats. Both sources have data about students including their* `id, city` *and* `courseNo` *of courses they registered for. The information at* s_1 *and* s_2 *satisfy the constraint* $id \rightarrow city$*. As we will see,* s_1 *and* s_2 *have some overlapping information.*

$source_1$

```
<university>
  <student>
    <ID>  100  <\ID>
    <name>  Ann  <\name>
    <course>
      <Course_No>  Mat444  <\Course_No>
      <Title>  Algebra   <\Title>
      <Credits>  3  <\Credits>
    <\course>
    :
    .
  <\student>
  <student>
    <ID>  120  <\ID>
    <name>  Chi  <\name>
    <city>  Greensboro  <\city>
    <course>
      <course_No>  Mat444  <\course_No>
      <Title>  Algebra  <\Title>
      <Credits>  3  <\Credits>
    <\course>
    :
    .
  <\student>
  :
  .
<\university>
```

$source_2$

```
<university>
  <course>
    <course_No> CSC555 <\course_No>
    <Title>   Data Mining <\Title>
    <Credits> 3  <\Credits>
    <student>
      <ID>   100  <\ID>
      <name> Ann  <\name>
      <city> Greensboro <\city>
    <\student>
    <student>
      <ID> 110  <\ID>
      <name> Bob  <\name>
      <city> Raleigh  <\city>
    <\student>
    :
    .
  <\course>
  :
  .
<\university>
```

The corresponding binary relations for these XML sources for `id-city` *and* `id-courseNo` *relations in the semantic-model view are shown below:*

source s_1

t_1

id	*city*
100	-
120	*Greensboro*

u_1

id	*courseNo*
100	*Mat444*
120	*Mat444*

source s_2

t_2

id	*City*
100	*Greensboro*
110	*Raleigh*

u_2

id	*courseNo*
100	*CSC555*
110	*CSC555*

Now consider the following query that lists the number of students from Greensboro in each course.

```
select   id-courseNo.courseNo, count(id-city.id)
from     id-city, id-courseNo
where    id-courseNo.id = id-city.id and
         id-courseNo.city = 'Greensboro'
group by id-courseNo.id .courseNo
```

We execute this query using all approaches introduced before. The materialization approach is the basic techniques against which other algorithms are evaluated. We will show in our example that the answers generated by other algorithms are exactly the same as the answers generated by the materialization technique. This is an indication of the correctness of our algorithms.

A. Materialization approach

1. For each source, we enforce the condition `city= 'Greensboro'` *to obtain:*

source s_1

t_1

id	*city*
120	*Greensboro*

u_1

id	*courseNo*
100	*Mat444*
120	*Mat444*

source s_2

t_2

id	*City*
100	*Greensboro*

u_2

id	*courseNo*
100	*CSC555*
110	*CSC555*

2. Send the results in each source to the mediator.

3. In the mediator, tables for the same relation are unioned (combined) at the mediator with duplicate removal. So, t_1 and t_2 are combined - with duplicate removal (let's call the result t), and u_1 and u_2 are combined - with duplicate removal (let's call the result u).

t

id	*city*
120	*Greensboro*
100	*Greensboro*

u

id	*courseNo*
100	*Mat444*
120	*Mat444*
100	*CSC555*
110	*CSC555*

4. In the mediator, query is executed to get the final answer. That will gives us:

course	*count(id)*
Mat444	*2*
CSC555	*1*

B. **Subqueries approach**

1. Subquereis phase: Generate and execute local and inter-source subquereis. Note that in this case the subqueries should not execute the aggregation since sources are overlapping. This will gives us (regardless where the inter-source subqueries are executed).

from local subquery at s_1 using t_1 and u_1

id	*city*	*courseNo*
120	*Greensboro*	*Mat444*

from local subquery at s_2 using t_2 and u_2

id	*city*	*courseNo*
100	*Greensboro*	*CSC555*

from inter-source subquery using t_2 and u_1

id	*city*	*courseNo*
100	*Greensboro*	*Mat444*

The inter-source subquery using t_1 and u_2 does not generate any answers.

2. Collect and merge phase: In the mediator, we union the previous results.

id	*city*	*courseNo*
120	*Greensboro*	*Mat444*
100	*Greensboro*	*CSC555*
100	*Greensboro*	*Mat444*

3. Additional processing phase: In the mediator, we apply the user query on the previous results to get the final answer.

courseNo	*count(id)*
Mat444	*2*
CSC555	*1*

C. **Wrapper approach**

1. In the wrapper approach, we will get the following relation at the mediator:

id	*city*	*courseNo*
100	-	*Mat444*
120	*Greensboro*	*Mat444*
100	*Greensboro*	*CSC555*
110	*Raleigh*	*CSC555*

2. We apply the chase for the functional dependency id → city on the previous results and remove the duplicate tuples.

id	*city*	*courseNo*
100	*Greensboro*	*Mat444*
120	*Greensboro*	*Mat444*
100	*Greensboro*	*CSC555*
110	*Raleigh*	*CSC555*

3. apply the user query.

courseNo	*count(id)*
Mat444	*2*
CSC555	*1*

CHAPTER VII
CONCLUSIONS AND FUTURE WORK

Information integration and interoperability among multiple information sources have been active research areas for more than a decade. More recently, due to increased application of XML for data representation, storage, and transmission, as well as significant increase in the volume of available information in electronic form, these problems have become even more significant and have received increased attention.

In this thesis we considered query processing algorithms developed in the *semantic model* approach to information integration, and extended these algorithms for queries involving aggragation and group-by operations. These algorithms include *materialization* (a centralized algorithm based on the creation of the semantic-model view), *subqueries* (a distributed algorithm based on dispatching subqueries to be executed at the information sources), and *wrapper* (a hybrid algorithm where information sources and the central coordinator participate in the execution of user queries). Our emphasis has been on developing extensions to these algorithms for the *efficient* processing of queries involving aggragation and group-by.

BIBLIOGRAPHY

[1] Dongfeng Chen, Rada Chirkova, Maxim Kormilitsin, Fereidoon Sadri, and Timo J. Salo. Pay-as-you-go information integration: The semantic model approach. 2007.

[2] Dongfeng Chen, Rada Chirkova, Maxim Kormilitsin, Fereidoon Sadri, and Timo J. Salo. Query optimization in xml-based information integration. In *CIKM*, 2008.

[3] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Ion Muslea, Andrew Philpot, and Sheila Tejada. The ariadne approach to web-based information integration. *International Journal of Cooperative Information Systems*, 10(1-2):145–169, 2001.

[4] Laks V. S. Lakshmanan and Fereidoon Sadri. Interoperability on xml data. In *The SemanticWeb - ISWC*, page 146163, 2003.

[5] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 233–246, 2002.

[6] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.