SEMANTIC QUERY OPTIMIZATION: A DATA-DRIVEN APPROACH

by

ALIREZA SHANKAIE

B.Sc., University of Tehran, 1993

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER of APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Electrical and Computer Engineering)

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

November 2000

© Alireza Shankaie, 2000
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical & Computer Eng.

The University of British Columbia
Vancouver, Canada

Date Nov 22, 2000
Abstract

The emergence of very large database systems over the last two decades has raised serious needs for more efficient query processing schemes. One of the proposed techniques for query optimization is Semantic Query Optimization (SQO), which is the subject of this thesis. One thing that distinguishes this technique from others is that it doesn't deal with low-level operations of the file system (e.g. block access scheduling, low-level indexing, etc). Here the objective is to alter the syntax of a query, without changing its semantics, in such a way that makes the query more efficient. In other words, by creating alternative queries, we aim at finding the alternative execution plan, which leads to the shortest execution time.

We adopted a data-driven approach in the sense that we use the stored data to extract useful information (inference rules) that could be used later by a query processor to construct alternative queries. The rules are stored in a relational format in files that are called meta-database. The query optimizer searches through these meta-databases for relevant rules.

We introduce the techniques that we have deployed to perform semantic query optimization in regards with our specific application. The results of experiments, as well as the amount of improvement achieved in each technique, are presented.
Table of Contents

Abstract ................................................................. ii
Acknowledgement ....................................................... vii
Table of Contents ...................................................... iii
List of Tables ............................................................ v
List of Figures ........................................................... vi

Chapter 1  Introduction ............................................... 1
Chapter 2  Problem and Background ................................ 5
  2.1 Semantic Query Optimization ................................. 5
Chapter 3  Related Work ............................................... 14
  3.1 Heuristic-based vs. Exhaustive Approach ................. 14
  3.2 Automatic Rule Derivation ..................................... 16
    3.2.1 Data-Driven vs. Query-Driven Approach .............. 17
Chapter 4  Application ............................................... 20
  4.1 Methodology ...................................................... 23
Chapter 5  SQO Techniques for Title Keywords Search ........ 27
  5.1 Introduction ...................................................... 27
  5.2 Title-keyword/Year Technique ............................... 27
    5.2.1 Building the Meta-database ............................. 30
    5.2.2 Experiment Results ....................................... 31
  5.3 Title Pair/Year Technique .................................... 33
    5.3.1 Low-AND-High Pairs ...................................... 37
        5.3.1.1 Experiment Results ................................. 39
    5.3.2 Low-OR-High Pairs ....................................... 42
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>A sample record in the Title-Keyword/Year meta-database</td>
<td>29</td>
</tr>
<tr>
<td>Table 2</td>
<td>Execution times for Title-Keyword queries</td>
<td>31</td>
</tr>
<tr>
<td>Table 3</td>
<td>Sample records in Title-Keyword meta-database</td>
<td>35</td>
</tr>
<tr>
<td>Table 4</td>
<td>Execution times for Low-AND-High keyword pair queries</td>
<td>40</td>
</tr>
<tr>
<td>Table 5</td>
<td>Execution times for Low-OR-High keyword pair queries</td>
<td>43</td>
</tr>
<tr>
<td>Table 6</td>
<td>Execution times for Low-AND-Low keyword pair queries</td>
<td>48</td>
</tr>
<tr>
<td>Table 7</td>
<td>Execution times for Low-OR-Low keyword pair queries</td>
<td>50</td>
</tr>
<tr>
<td>Table 8</td>
<td>A sample tuple in the Author-Year meta-database</td>
<td>53</td>
</tr>
<tr>
<td>Table 9</td>
<td>Execution times for Author queries</td>
<td>57</td>
</tr>
<tr>
<td>Table 10</td>
<td>Execution times for Author-Title keyword pair queries</td>
<td>62</td>
</tr>
<tr>
<td>Figure 1</td>
<td>A sample record in the citation database</td>
<td>20</td>
</tr>
<tr>
<td>Figure 2</td>
<td>The general view of the implementation of SQO techniques</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3</td>
<td>A general view of Title-Pair/Year technique</td>
<td>37</td>
</tr>
<tr>
<td>Figure 4</td>
<td>The pseudo-code for the Low/High sample-creating algorithm</td>
<td>39</td>
</tr>
<tr>
<td>Figure 5</td>
<td>How the Author/Year rules are formed</td>
<td>56</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Author/Year (Section 6.2)</td>
<td>75</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Title keyword/Year (Section 5.1)</td>
<td>75</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Low-AND-High Pair (Section 5.2.1)</td>
<td>76</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Low-AND-High Pair (Section 5.2.2)</td>
<td>76</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Low-AND-Low Pair (Section 5.2.3)</td>
<td>77</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Low-OR-Low pair (Section 5.2.4)</td>
<td>77</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Author-Title/Year (Section 6.3)</td>
<td>78</td>
</tr>
</tbody>
</table>
Acknowledgement

I would like to thank my supervisor, Dr. Babak Hamidzadeh, for his valuable support during my studies. Without his spectacular support and encouragement, as a teacher and a friend, especially at the beginning of my studies, I would have not been able to complete this successful journey.
1 Introduction

The emergence of very large database systems over the last two decades has raised serious needs for more efficient query processing schemes. Apart from the ever-increasing size of databases, the deployment of the world-wide-web as a means of remote data delivery to and access by a very large number of users has increased the importance of efficient query processing techniques even more.

Such techniques optimize the execution of queries imposed on the database in terms of cost and time. The notion of cost may involve the time factor as well, since time can affect the cost of execution of the query. However, in this thesis like other works in the literature, we consider time as a factor that can be considered independently for optimization purposes. Here, we simply assume that the less the time of execution of a query, the more efficient the query becomes. Other factors, such as memory space, are not a concern to us in this research.

One of the proposed techniques for query optimization is Semantic Query Optimization (SQO), which is the subject of this thesis. It was first introduced by John King in the early 80's [KING1] [KING2]. One thing that distinguishes this technique from others is that it doesn't deal with low-level operations of the file
system (e.g. block access scheduling, low-level indexing, etc). Here the objective is to alter the syntax of a query, without changing its semantics, in such a way that makes the query more efficient. In other words, by creating alternative queries, we aim at finding the alternative execution plan, which leads to the shortest execution time. Note that we do not compromise the accuracy of the results, meaning that we would like to retrieve the same set of data when executing an alternative query, as when we would be using the original query.

The SQO techniques have not caught the attention of the commercial system developers as much as they should. There might be several reasons for that. The most prominent one is the fact that SQO was in many cases designed for deductive databases and because of this association, SQO might not appear useful for relational databases. Second, at the time when SQO techniques were being developed, the relative CPU and I/O speeds were not as dramatically different as they are now. The saving in query execution time (dominated by I/O) that SQO could provide was not worth the extra CPU time necessary to optimize a query semantically. And finally, there has been the impression that in order to take advantage of SQO techniques, it is essential to extract all the inference rules that may exist in the database. We argue that this is not always the case. The simple reason for that is because of the higher speeds of CPUs and I/O devices today, the cost of optimization techniques can be negligible. So, even if we apply the optimization techniques to all the received queries and even if we
already know that only 50% of the queries can be practically optimized, the overhead imposed by the other 50% will still constitute a negligible amount.

In order to be able to form more efficient queries, we need to use a set of inference rules. Inference rules depict the useful relationships between attributes of the database. In some applications, like ours, rule extraction requires domain analysis. We applied SQO techniques to a citation database that stores bibliographical information. The database is in textual format.

We adopted a data-driven approach in the sense that we use the stored data to extract useful information (inference rules) that could be used later by a query optimizer to construct alternative queries. The rules are stored in files that are collectively called the meta-database. The query optimizer searches through the meta-database for relevant rules.

This thesis is organized as follows. Chapter 2 presents the problem and background on the subject. Chapter 3 covers the related work done by other researchers in the area. Chapter 4 introduces the system and application on which we carried out our experiments and describes the differences between our system and relational database systems. In Chapters 5 and 6, we propose the possible techniques for implementing SQO with regards to the specifications of search operations (submitted queries). Each of these two chapters covers the SQO technique for a specific class of queries. Each chapter also, includes the
results of our experiments for each technique. In Chapter 7, we address the issue of updates with regards to the specification of our proposed SQO module. We summarize our work in Chapter 8 and describe future extensions of the work.

Throughout this thesis, words “tuple”, “field” and, “attribute” are used interchangeably. Also, words “table” and “relation” refer to the same entity. For sample queries, a standard SQL-like notation has been used. Also, the words representing the attributes (Author, Title, etc) in the database are printed with the first letter in capital.
Problem and Background

In this chapter we investigate the main concepts in semantic query optimization. We will mainly focus on techniques that have been proposed by the researchers in this field.

2.1 Semantic Query Optimization

Semantic query optimization aims at forming a set of alternative queries to an original query and then finding the most efficient query among the alternative queries. All the alternative queries are semantically equivalent but syntactically different. Having different syntax means removing those parts of the query that are redundant or adding new elements that are useful.

Semantic query optimization uses knowledge about possible database states to reduce query execution time. The knowledge is stated as integrity constraints, such as functional dependencies and bounds on numeric attribute values. For example, the integrity constraint “The size of classes is always less than 200” can be used to optimize a query that queries the number of classes with capacity more than 250. In this case, there is no need to access the database. This is just a sample case where no access to the database is required. In fact, in most of
the cases the query processor still needs to access the database but it takes much shorter time to execute the query.

One important issue here is that sometimes the integrity constraint is static and is enforced from the beginning (by the database architect for example). Such constraints will not change over time. There are, however, other cases where the constraint (rule) is dynamic and can change from time to time. These types of rules should be discovered from within the database and stored in a meta-database and an updating process should be carried out on a regular basis to keep the meta-database up-to-date.

In order to transform a query to a more efficient one by changing its syntax, a number of techniques are often used [SIEGEL] [GRYZ]. One is constraint introduction (addition) or removal and another one is join removal. Furthermore, sometimes we can find the answer to a query without even accessing the database. It can be either because the answer set is empty or the answer is available in the query itself (tautology). We describe these techniques through the remaining of this section.

- **Constraint Introduction/Removal**

Considering the above explanation, semantic query optimization, in a sense, is addition or removal of constraints to a query in accordance with a given rule set.
The transformed query is semantically equivalent to the original query in that, regardless of the database state, the same answer will be returned from the database. Let $Q$ be any query expressed as a conjunction of constraints,

$$Q = C_1(A_{i1},...,A_{ik}) \land \ldots \land C_n(A_{n1},...,A_{nk})$$

Where $C_1(A_{i1},...,A_{ik})$ is a constraint on attributes $A_{i1},...,A_{ik}$ and $C_n(A_{n1},...,A_{nk})$ is a constraint on attributes $A_{n1},...,A_{nk}$, where the two sets of attributes need not be disjoint. Then given a rule of the form,

$$C_i(A_{i1},...,A_{ik}) \land \ldots \land C_j(A_{j1},A_{jk}) \Rightarrow C_i(A_{i1},...,A_{ik}) \land \ldots \land C_m(A_{m1},...,A_{nk}),$$

we can transform query $Q$ if one of the following methods apply:

**Constraint introduction**

If a subset of the constraint in the query implies the antecedent of a rule then the consequent of that rule can be added to the query [SIEGEL]. If evaluating the antecedent of the implication over any database state returns a subset of the relation returned by evaluating the consequent over the same database state, a set of constraints is said to imply another set of constraints,
\[ C_i(A_{i1}, \ldots, A_{ik}) \land \ldots \land C_j(A_{j1}, A_{jk}) \Rightarrow C_i(A_{i1}, \ldots, A_{ik}) \land \ldots \land C_m(A_{m1}, \ldots, A_{mk}). \]

As an example of constraint introduction, given the following query,

\[ Q_i = C_1(A_{11}) \land C_2(A_{21}) \]

And the inference rule:

\[ \text{Rule}_k: C_1(A_{11}) \Rightarrow C_3(A_{31}) \]

We can transform \( Q_i \) into the equivalent query \( Q_j \).

\[ Q_j = C_1(A_{11}) \land C_2(A_{21}) \land C_3(A_{31}) \]

Constraint introduction can improve the performance in two ways. One is by introducing a new constraint on an attribute, which is indexed. The conventional query optimizer will then be able to take advantage of the indexed attribute to retrieve the matching tuples faster. The second method is introducing a range constraint that reduces the scan overhead of the database and results in shorter processing times.

Applying the above method doesn't always result in an optimized query. Sometimes the cost of executing a query with extra redundant constraints can be
considerably higher. It is a major task of a Query Optimizer process to predict the cost of transformed queries by means of heuristics or experience.

*Constraint Removal*

If a subset of the constraints in the query implies the antecedent of a rule and the consequent of that rule implies a subset of the constraints in the query, then the constraints implied by the consequent of the rule may be removed from the query [SIEGEL]. As an example of constraint removal, we can simply work backwards from $Q_j$ to $Q_i$ using Rule$_K$.

Consider the original query and Rule$_K$ as follows:

$$Q_j = C_1(A_{11}) \land C_2(A_{21}) \land C_3(A_{31})$$

Rule$_K$: $C_1(A_{11}) \Rightarrow C_3(A_{31})$

Here both the antecedent and consequent of the rule are implied by the query so based on the above explanation we can remove the consequent of the rule ($C_3(A_{31})$) from the query and still keep the semantics of the query intact. The transformed query will be:

$$Q_i = C_1(A_{11}) \land C_2(A_{21})$$
Join operations in queries are very costly. Traditional query optimizers use a number of techniques (namely join re-ordering) to reduce the cost of join operations, however, they never attempt to eliminate the redundant joins. In semantic query optimization, by using inference rules, a redundant join operation can be detected and removed hence creating a more efficient query. This could lead to significant time savings [GRYZ]. We explain the join removal method using the following example:

Suppose that we have two tables in our database. One stores the personal information of employees (Emp_Info) and the other one stores the information about who works in what project (Emp_Project). One employee can be involved in more than one project at the same time. The primary key for both tables is SIN. Suppose we need to retrieve a list of all employees who are participating in the project “SQO” and are holding PhD’s. The original query can be like the following:

```
Select Emp_Info.Name From Emp_Info, Emp_Project
Where Emp_Info.SIN = Emp_Project.SIN AND
Emp_Project.Name = “SQO” AND
Emp_Info.Degree = “PhD”
```
This original query requires a join operation between the two tables. However, we can remove the Join if the following rule holds:

\[ \text{Emp\_Info.Degree} = \text{"PhD"} \Rightarrow \text{Emp\_Project} = \text{"SQO"} \]

It means that all the employees with a PhD degree are participating in project “SQO”. Considering the above rule, we will no longer need the join operation. The transformed – and more efficient – query will be as the following:

Select Name From Emp\_Info Where Degree = “PhD”

As can be seen the optimized query does not need to access the table Emp\_Project, which results in considerable time saving. In our application we have only one table in the database so we did not consider the join removal technique.

- Detecting the Empty Answer Set and Tautology

Sometimes it is possible to find the answer to a query without even accessing the database [SIEGEL] [GRYZ]. One case is when there is a constraint in the query that contradicts one or more constraints of a rule. This contradiction can be permanent (regardless of the current state of the database) or temporary (dependent on the current state of the database).
Consider the following Query:

Select Employee_Name where Employee_Age = 25 AND Employee_Salary > $60,000

If there is a rule in our meta-database such as:

Employee_Age < 30 ⇒ Employee_Salary < $55,000

Since one of the constraints in the query (Employee_Age = 25) is a subset of the antecedent of the rule, then we can transform the query into the following:

Select Employee_Name where Employee_Age = 25 AND Employee_Salary > $60,000 AND Employee_Salary < $55,000

There are two constraints in the query that contradict each other. Then we can conclude that the answer to the submitted query is Null.

This kind of semantic optimization can result in significant time saving (close to 100%). However, the frequency of facing such a query is not high. We will see that in our system we have taken advantage of this technique.
The second way in which a query can be answered without accessing the database is if the answer is present in the query itself. This method is called *Tautology*. As an example, suppose the following rule holds:

Employee\_Position = "manager" \(\Rightarrow\) Employee\_Salary \(\geq\) $80,000

If the submitted query is:

\[
\text{Select min(Employee\_Salary) where Employee\_Position = "manager"}
\]

Then using the above rule, the query can be transformed to:

\[
\text{Select min(Employee\_Salary) where Employee\_Position = "manager" AND Employee\_Salary \geq $80,000}
\]

The answer can be extracted directly from within the query. In this case, it is possible to reach time saving close to 100% provided that the overhead for query transformation is negligible.
3 Related Work

One issue that has been the subject of extensive research in the area of semantic query optimization is how to acquire the knowledge required for SQO techniques described in Chapter 2. This is a stage that is called learning stage or knowledge discovery stage. In this chapter, we summarize the work of other researchers who have tackled this issue from different points of view.

3.1 Heuristic-based vs. Exhaustive Approach

When the query optimizer attempts to form all the possible alternative queries to an original query (transformation phase), it should inspect the rule set and extract all useful rules. One approach is to check every single rule that is contained in the rule set and determine whether it can be used for optimization (exhaustive search). This approach turns out to be very inefficient specially when the rule set is very large. In order to address this issue, researchers [SIEGEL] [KING1] [HAMMER] [FREYTAG] [GRAEFE] have proposed some general heuristics that can be used by the query optimizer during the transformation phase. These heuristics help the query optimizer to check only a part of the rule set.

In the heuristic-based model, a proven set of transformation heuristics is used to determine which rules can be used to transform a given query into one that is less costly to process. One example of heuristics is:
If there is a constraint $C_1$ on attribute $A_1$ of relation $R$ in a query and there exists an index on attribute $A_2$ of relation $R$, then try to find a rule in which $C_1$ appears in the antecedent and a constraint $C_2$ appears in the consequent.

Regardless of the initial approach, the optimizer always accesses the rule set at some point. Rule set is an ever-changing entity specially when the data change as time passes. The rules may be fixed during the database lifetime or may change as a result of database updates. For example, the rule:

\[
\text{City\_Location} = \text{"Canada"} \Rightarrow \text{Distance\_from\_London} \geq 3000 \text{ km}
\]

Will always remain valid and is considered as a static rule. (No city in Canada is located closer than 3000 km away from London). However a rule such as:

\[
\text{Employee\_Age} < 30 \Rightarrow \text{Employee\_Salary} < 60,000
\]

may change when a new employee younger than 30 is hired with a salary above $60,000.
3.2 Automatic Rule Derivation

In the early stages of SQO research, the query transformation was performed based on a set of static semantic integrity constraints. These constraints need to be specified explicitly by users and/or database administrators. In many applications it is very difficult to identify all relevant integrity constraints. Additionally, some of the constraints specified by the user or DBA have only significance for consistency checking and not for SQO purposes. One example of such a constraint is:

\[ \text{Employee\_Salary} > \$0 \]

Because of this limitation, researchers proposed methods for Automatic Rule Derivation.

The process of automatic rule derivation is known in the field of Artificial Intelligence as inductive learning [SIEGEL]. Some researchers have addressed the question of how heuristics can be used to derive knowledge from data or operations on data. We have used automatic rule derivation in our system. Heuristics were not used in our system, because of the nature of the system that makes it possible for us to come up with an accurate and self contained dynamic meta-database that includes all rules. This is one of the differences of our system compared to the systems considered in previous researches by others.
Again, as a result of the unique specification of our system, we can ensure that if there is any rule in the meta-database, it is a useful one. In fact, in our system the rule verification and validation process is quite simple.

[HSU] has investigated inductive learning methods for generating rules using the submitted queries. The system \textit{gradually} learns a set of effective rules for optimization. Time saving up to 25\% has been reported.

3.2.1 Data-Driven vs. Query-Driven Approach

There are two general approaches for automatic rule derivation. One is referred to as the \textit{data-driven} [SHEKHar] in which the contents of the database (tuples) are examined to extract the useful inference rules that can be used later by the semantic query optimizer.

The advantage of this approach is that, we are, potentially, able to derive all existing rules in the database, which results in an accurate and comprehensive set of rules. Of course, there is the possibility of deriving those rules that are valid but not useful for our optimization purposes. One of the reasons for that might be that the query processor never receives a query that can be optimized using those rules. This case suggests a different approach, referred to as \textit{query-driven}.
In the query-driven approach, stochastic examinations are made to grasp
knowledge about the nature of the queries submitted by users [YU]. The process
responsible for rule extraction can use a log of the queries submitted so far to
determine if it is worthwhile searching for any possible rule with a specific
attribute appeared in its antecedent or consequent. For example, suppose that
the query log shows that in less than 2% of the cases the users were querying
about the attribute Employee_SIN. Then, the system can just refrain from
attempting to discover any rules associated with this attribute.

The immediate disadvantage of this approach is that, as new queries arrive over
time and especially when the needs of the users shift to a different area for
whatever reason, the accuracy and comprehensiveness of the rule set
diminishes. Note that the deterioration is not a result of adding more tuples to the
database. It may happen even if there is absolutely no change in the state of the
database.

Another interesting implication of the query-driven approach is when the answers
to two completely different queries happen to be exactly the same. The following
example describes this situation better:

Suppose the results of the following two queries turn out to be identical.

Q1: Select * from Emp_Info where Employee_Age < 32
Q2: Select * from Emp_Info where Employee_Degree = “BSc”

As can be seen, the two constraints introduced in the queries are totally different. But because of the identical answer sets we can conclude the following rules:

R1: Employee_Age < 32 ⇒ Employee_Degree = “BSc”
R2: Employee_Degree = “BSc” ⇒ Employee_Age < 32

Note that the identical answers of the above queries imply that All Employees who are 32 or younger have a B.Sc. degree and All Employees with a B.Sc. degree are 32 or younger. In other words, we can not find an employee who is 40 and with a B.Sc. degree. Nor can we find an Employee who has a Masters degree and is younger than 32.

Now, suppose the database is indexed on attribute Employee_Degree. In that case, if we face a query that contains a constraint on Employee_Age, then we might be able to take advantage of R1 and introduce an additional constraint with an indexed attribute which is expected to make the query faster to execute.

We have used the data-driven approach in our project. However, the query-driven approach is also applicable. A combination of both approaches can also result in good improvements.
4 Application

Our system is a textual database for scientific and engineering citation information. The data are stored in text files in SGML format. This database is used as a part of a larger commercial system. Since the database has a textual format, each line of text in data files corresponds to a record. Fields (attributes) in each record are identified by special tags. Some of the attributes that are of higher importance for our purposes are Title, Author and, Year. Some other available attributes are journal name, bibliographic information (that includes volume number and page numbers, etc) and, a unique key tag. The total number of tuples (records) in the system is about 23,000,000. The following is a sample tuple in the citation database:

```
```

**Figure 1:** A sample record in the citation database

The sample record stores the information for an article authored by J. Benson and D. Cooper. Tags <AU> and </AU> denote the beginning and end of attribute "Author". Tags <TI> and </TI> mark the beginning and end of attribute "Title". These two attributes are the core of different classes of queries that we have
considered in this research. <S> and </S> show the beginning and end of attribute Source (journal, book, etc). <Y> and </Y> represent the attribute Year. <BIB> and </BIB> represent the bibliographical information and finally <UKEY> and </UKEY> represent a unique key that is automatically created by software while entering the information into the citation database.

Our SQO methods mainly work on Author, Title and, Year attributes. These are the fields that most of the times appear in user queries. The optimizer is not likely to receive a query with a constraint on fields such as volume number or page number. Although the original system without SQO features can, in general, deal with that kind of queries as well

Another subtle issue is the way the data files are organized. Data are not stored in one single large database. In fact, the entire database is organized as separate files for each year. The year range spans over a 26-year period from 1974 to 1999. These files are located in separate physical and logical locations and have exactly the same structure and are completely isolated from each other meaning that Join operations have no significance in our system. This, of course, means that the SQO techniques regarding join elimination are not applicable in our system. Instead, we focus on other techniques, namely constraint introduction.
The primary key in the database is the "Author" field. So one of our objectives could be to take advantage of the existing index file on Author. This can be done by introducing constraints on "Author" to a query to make it more efficient. By conducting manual experiments, we realized that introducing constraints on the Author attribute does not result in significant performance improvement. Instead, we focused on introducing constraints on the Year attribute. If the Year constraint is not present in the original query, the query processor, by default, searches all the years for the matching tuples. By adding the Year constraint to the original query, the processor only accesses the relevant year files and as a result, executes faster.

One of the advantages of SQO techniques is that it does not involve itself with low-level file operations as is the case with conventional query optimizers. It acts as an intermediate stage between the front end and the query processor (or database engine). In our system, the SQO module first examines and probably transforms the queries and then passes it onto the query processor to be executed. So, we look at the query processor as a Black Box and during the course of the project do not get involved in the internal operations of this program. The transformation phase takes advantage of a set of relations in a meta-database that contains useful information about the data. We categorized the queries into different classes based on the technique that can be applied. These techniques will be introduced in full detail in the
subsequent chapters. A distinct component of the meta-database is used for a specific technique.

4.1 Methodology

Our method of optimizing the query execution is by creating and maintaining a meta-database that represents the relationships between fields in our database. A query Q is first passed onto the query optimizer. The output from the query optimizer is a semantic equivalent of Q, namely Q', which is then passed onto the query processor for processing, and is expected to execute more efficiently than Q. The meta-database itself is indexed for efficient execution.

Our approach to query optimization is data-driven in the sense that we derive the relationships between database fields from their values in the current snapshot of the database. Thus, the meta-database is first created by searching the database for such relationships, extracting the relationships from the database and then storing them in the meta-database. A change in the current state of the database may create a mismatch between it and the relationships that are represented in the meta-database. This prompts updating the meta-database upon database updates. We will discuss the issue of update in more detail in Chapter 7.

We have more than one meta-database, one for each technique described later in the next chapters. In some cases, it might be required to access more than a
single meta-database in order to form the optimized query. Our measurements revealed that the time overhead caused by accessing the meta-database to retrieve the rules is negligible.

And finally, it is important to note that in our implementation of SQO a number of techniques have been developed and there is one distinct component of the meta-database associated with each technique. The optimizer inspects the query and determines which technique, if any, is appropriate for the submitted query. The meta-database is then accessed to retrieve any possible useful rule. We regard each record in the meta-database as a rule and by using the word rule throughout this thesis, we refer to a record in the meta-database. This definition distinguishes the concept of rule from what is usually considered in the research literature. We will explain this matter in Chapter 5 when we are describing each SQO technique in more detail.

In order to give the reader a visual aide of how the system as a whole works, Figure 2 is presented that shows the flow of operation in the system. The SQO engine that acts as a mediator between outside world and the query processor receives an original query. The SQO module should first determine which of the techniques suits the submitted query. Depending on the selected technique, a meta-database is scanned for extracting any possible useful rules, and the rules are returned to the SQO engine to transform the original query to a semantically equivalent but more efficient query. The names in the ovals denote the
techniques that have been considered for this project. In order to avoid increasing the visual complexity of the figure, only three techniques have been explicitly presented.

**Figure 2:** The general view of the implementation of SQO techniques
In the next few chapters, we describe the techniques that we have adopted, with respect to our specific system, for performing semantic query optimization. Each technique will be described in detail and the results of our experiments will be presented. Before that, it is important to consider how the query processor, without any optimization, manipulates submitted queries.

The original query processor, without SQO features, requires that a Year constraint be present in all queries. If a query does not explicitly specify a Year constraint, the query processor adds a Year constraint to the query that contains all the years for which the information is stored in the database (from 1974 to 1999). That requires accessing all files that store the information, since the organization of data is based on one file for each year. This is obviously very inefficient and the main objective in our techniques, was to eliminate the redundant years from the Year constraint. Throughout this thesis, we refer to a Year constraint that spans over all years as *full range*.
5 SQO Techniques for Title Keyword(s) Search

5.1 Introduction

In this chapter, we assume that the submitted queries contain either one constraint on Title attribute or two constraints on a pair of Title attributes. Initially, no Year constraint is present in the query. This is the main task of the optimizer to identify the relevant Year constraints and append them to the original query.

5.2 Title-Keyword/Year Technique

One of the attributes in the textual database that has significant importance for SQO purposes is "Title". The reason is that it is very common for users to search for a paper or an article according to a keyword that appears in the title. We can take advantage of the fact that some keywords appearing in the "Title" field can be found only in few years. By extracting those years and introducing a new constraint based on Year field, we can reduce the range of years that should be scanned while executing the query.

Before creating any meta-database, we should consider the fact that there are many frequently used words that appear many times in the titles and in every
year in the range. These keywords, High-Frequency keywords as we call them, are not proper candidates for this technique. For example, if a query were supposed to extract all the records containing the keyword “Computer”, using this technique would not yield a performance improvement since the associated year range for the keyword “Computer” covers all the years in our database. So, searching the meta-database to extract the rule associated with this keyword would not be a logical exercise, thus including this keyword in the meta-database in the first place is not logical either.

The above explanation prompts us to the notion of High-Frequency vs. Low-Frequency keywords. While creating the meta-database for this technique, we need to identify High-Frequency keywords and filter them out of the meta-database. A rough definition for a High-Frequency keyword is a word for which a search query on all years, returns at least one hit for each year in the range. However, if a search on a keyword returns, let’s say, 100 hits but in only 10-year range, it will still be considered as a Low-Frequency keyword since our first priority is to shrink the year range regardless of the number of hits for the reduced year range. Some of the obvious High-Frequency keywords (usually known as stop words) were left out without even accessing the database to count their occurrences. A list of such keywords was created by hand and included words such as “the”, “for”, “and”, and a number of others. It turned out that most of the keywords that were identified as Low-Frequency were indeed scientific terms such as the names of chemical components, for example “D-
GLUCOPYRANOSIDE", or new terms that have recently entered the literature, for example “CORBA”.

Based on the above discussion, a record in the meta-database for Keyword/Year technique can have the following form:

<table>
<thead>
<tr>
<th>Title-Keyword</th>
<th>Years</th>
</tr>
</thead>
</table>

Table 1: A sample record in the Title-Keyword/Year meta-database

It means that all the records that contain the word “CORBA” in their title field can be found only in the specified years. This rule doesn’t say anything about the number of records in each specific year that contain the word “CORBA” in its title field. For example, there might be –and actually is quite likely- 200 records only in 1999 that contain the word “CORBA” in the title. This doesn’t reduce the usefulness of the above rule because we can still avoid scanning years (scanning files in every year directory) that don’t have any records about CORBA and because of the specific structure of our database, it will save us considerable amount of time. So if the query processor faces a query like:

Select ... Where Title contains “CORBA”,

the query optimizer can transform it to a more efficient query like:
Select ... Where title contains "CORBA" AND

(Year = 1993 OR Year = 1994 OR Year = 1995 OR Year = 1998
OR Year = 1999)

5.2.1 Building the meta-database

In order to build the meta-database for Title/Year technique, we divided the entire process into separate sub-processes, based on year. In each sub-process, we identified all the words that had appeared in the Title field of records in each examined year. All the stop-words were filtered out. An output file was created for each year containing all the words that could potentially be a Low-Frequency keyword. All these files were sorted and merged together to create a large file containing all candidate words along with their corresponding years. At this stage, the second round of filtering out was carried out. If a word was identified to have a full range of corresponding years, it was considered as a High-Frequency keyword and was filtered out. Otherwise, it was stored in the final Title/Year list. Finally, the Title/Year list was indexed. The index file was, in fact, the meta-database. The total number of records in the final rules table is close to 9,200,000.
5.2.2 Experiment Results

Based on the above technique, we performed experiments using a large number of queries with a single Title Keyword specified. 11,000 specific Low-Frequency keywords that appeared in the search were randomly selected from the citation database. Table 2 presents the results of this experiment.

<table>
<thead>
<tr>
<th>Number of Year constraints</th>
<th>Number of cases</th>
<th>Optimized (sec)</th>
<th>Non-Optimized (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3764</td>
<td>.2234</td>
<td>.3856</td>
</tr>
<tr>
<td>2</td>
<td>1478</td>
<td>.2769</td>
<td>.4307</td>
</tr>
<tr>
<td>3</td>
<td>989</td>
<td>.3358</td>
<td>.4843</td>
</tr>
<tr>
<td>4</td>
<td>714</td>
<td>.3854</td>
<td>.5206</td>
</tr>
<tr>
<td>5</td>
<td>541</td>
<td>.4485</td>
<td>.5642</td>
</tr>
<tr>
<td>6</td>
<td>489</td>
<td>.5156</td>
<td>.6101</td>
</tr>
<tr>
<td>7</td>
<td>379</td>
<td>.5150</td>
<td>.6157</td>
</tr>
<tr>
<td>8</td>
<td>319</td>
<td>.5615</td>
<td>.6715</td>
</tr>
<tr>
<td>9</td>
<td>275</td>
<td>.6089</td>
<td>.6880</td>
</tr>
<tr>
<td>10</td>
<td>217</td>
<td>.6938</td>
<td>.7413</td>
</tr>
<tr>
<td>11</td>
<td>229</td>
<td>.7789</td>
<td>.8604</td>
</tr>
<tr>
<td>12</td>
<td>192</td>
<td>.8407</td>
<td>.8629</td>
</tr>
<tr>
<td>13</td>
<td>156</td>
<td>.7828</td>
<td>.8529</td>
</tr>
<tr>
<td>14</td>
<td>149</td>
<td>1.086</td>
<td>1.107</td>
</tr>
<tr>
<td>15</td>
<td>142</td>
<td>1.029</td>
<td>.9974</td>
</tr>
<tr>
<td>16</td>
<td>112</td>
<td>1.199</td>
<td>1.190</td>
</tr>
<tr>
<td>17</td>
<td>96</td>
<td>1.377</td>
<td>1.317</td>
</tr>
<tr>
<td>18</td>
<td>90</td>
<td>1.789</td>
<td>1.733</td>
</tr>
<tr>
<td>19</td>
<td>102</td>
<td>1.491</td>
<td>1.450</td>
</tr>
<tr>
<td>20</td>
<td>80</td>
<td>1.616</td>
<td>1.566</td>
</tr>
<tr>
<td>21</td>
<td>81</td>
<td>1.652</td>
<td>1.600</td>
</tr>
<tr>
<td>22</td>
<td>74</td>
<td>1.913</td>
<td>1.801</td>
</tr>
<tr>
<td>23</td>
<td>81</td>
<td>1.975</td>
<td>1.915</td>
</tr>
<tr>
<td>24</td>
<td>82</td>
<td>2.302</td>
<td>2.107</td>
</tr>
<tr>
<td>25</td>
<td>97</td>
<td>2.586</td>
<td>2.433</td>
</tr>
<tr>
<td>26</td>
<td>72</td>
<td>2.919</td>
<td>2.685</td>
</tr>
</tbody>
</table>

*Table 2: Execution times for Title Keyword queries.*
The above experiment provided the following summarized results:

<table>
<thead>
<tr>
<th>Optimized time: 0.5077 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Optimized time: 0.6209 sec</td>
</tr>
</tbody>
</table>

This technique resulted in an average 18.3% improvement.

It is important to note that in this technique, the transformed query does not always perform better than the original one. In other words, we can notice a cross-over point around 15-years case. It means that from that point on executing the semantically equivalent query is not more efficient than the original query. This is not surprising considering the fact that there is a cost associated with the optimization process (i.e. accessing the meta-databases). Furthermore, from a quantitative point of view, this deficiency doesn't seem to be dramatic since most of the cases happen to be occurring for a smaller number of years. For example, out of a total of 11,000 sample keywords 3764 (34%) are 1-year cases and 1478 (13%) are 2-year cases and only 72 (less than 1%) are 26-year cases. That is why despite a cross-over point at around the 15-year point, we could still gain an overall improvement of more than 18%. The 26-year cases are those sample keywords for which no match was found in the meta-database. These were considered to be High-Frequency keywords.

Based on the above explanation, if we calculate the average times only for those cases that occur more frequently, we can expect greater performance improvement. For example, the following numbers represent the average times
for samples between 1-year and 10-year points. Out of the total of 11,000 samples, 9165 fall within this range.

Optimized time: .3320 sec  
Non-Optimized time: .4734 sec

The average improvement in this case is 29.8%. One interpretation of the above numbers could be that more than 83% of the users (9165/11000) experience an average performance improvement of close to 30%.

If we consider only the first five rows of Table 2 (cases between 1-year and 5-year points), the optimized and non-optimized average times will be as follows:

Optimized time: .2805 sec  
Non-Optimized time: .4333 sec

In this case, the average performance improvement is 35.2%.

5.3 Title Pair/Year Technique

It is quite common that users submit queries that contain more than one title keyword constraint. Sometimes it is in the form of an “AND” (conjunctive) statement, which narrows down the result or in the form of an “OR” (disjunctive) statement, which expands the result. As a result, in this project, in addition to single title keywords (Section 5.2), we decided to address the keyword pair cases as well.
Depending on two factors, the experiments were performed in four distinct categories. In other words the Title-Pair/Year technique is made up of four sub-techniques that are presented in this section. First, we discuss the factors that distinguish different sub-techniques from one another. Then the results of each sub-technique are presented along with explanations.

The meta-database used in this technique is the one that was used for Title-keyword/Year technique described in Section 5.2. The only difference is that here we need to scan the meta-database twice.

Suppose we have extracted the corresponding rule (years) for each keyword in the pair. Depending on the way the keywords are bounded in the query, the final year statement to be used in the optimized query is formed. We elaborate on the above statement by the following example.

Suppose the submitted query is as follows:

Select ... Where title_contains “CORBA” AND title_contains “JINI”

Also, suppose that after scanning the meta-database, we have extracted the following two rules for the keyword pairs:
Since the user wishes to retrieve all the records that contain both keywords in their title, the year constraint to be inserted into the transformed query would be the intersection of the above two year statements hence 1993, 1994, 1999. It is clearly evident that we have gained a twofold improvement by narrowing down the year scanning to those years that guarantee at least one match for both keywords. The final optimized query is like the following:

Select ... Where title_contains "CORBA" AND title_contains "JINI" AND

   (Year = 1993 OR year = 1994 OR Year = 1999)

Now, as a different example, consider the following query that binds the two keywords by OR:

Select * ... Where title_contains "CORBA" OR title_contains "JINI"

The optimizer goes through the same procedure as the previous example, except for when it attempts to form the Year constraint for the transformed query. The user wishes to retrieve all the titles that contain at least one of the two title keywords. So considering the same rules as in Table 3, the computed Year

It might be argued that for the OR case, the final Year constraint is larger than both Year constraints associated with each keyword. Then, is it going to degrade the efficiency of this technique? The answer is obviously No! It is important to remember that we are comparing the transformed query to the original one that implicitly forces the query processor to scan the whole range of 26 years. We can argue that no matter how big the "ORed" Year constraint becomes, it is still expected to be smaller than the full range. There is only one extreme situation in which the "ORed" Year constraint might become as big as the full range and that's when one of the Year constraints associated with a keyword happens to be the full range (26 years). But as we have mentioned earlier in section 5.1, we only store the rules for Low-Frequency keywords in the meta-database meaning that we may never retrieve a 26-year Year constraint for any keyword stored in the meta-database. This introduces the second factor that divides the main Title-Pair/Year technique into a number of sub-techniques as mentioned at the beginning of this section and that is the concept of Low-Frequency vs. High-Frequency keywords and their combinations.

Any pair of keyword constraints contained in a query can be Low/Low, High/High, or Low/High combination. The High/High combination is not of any interest to us since the Year-constraints for both of them are full range, hence a full range final
Year constraint for both "OR" and "AND" cases. That immediately suppresses any hope for getting an improvement for this kind of combinations. However, the other two combinations, namely Low/Low and Low/High, seemed to be appropriate candidates for SQO techniques and we carried out experiments to verify this speculation. The following figure presents the above description in a visual form:

![Diagram of Title-Keyword Pair/Year Technique](image)

**Figure 3:** A general view of Title-Pair/Year technique

Throughout the following sections, we present the details of each of the sub-techniques including the results of our experiments.

5.3.1 Low-AND-High Pairs

In all of the techniques that involve pairs, one issue is using effective methods for creating sample pairs for the experiments. In case of Low/Low pairs, we can
solely rely on the data that is available in the meta-database since all the Low-Frequency keywords are gathered in them. In case of Low/High pairs, the situation is a bit more complicated since we don't have any direct access to a repository of High-Frequency keywords except the citation database itself. Furthermore, it is important to form pairs that can return a match. In other words, the result of the sample query should not be null.

In order to address the above issues, we decided first, to select a Low-Frequency keyword randomly and use it to find a suitable High-Frequency keyword. The pseudo-code in Figure 4 presents the implemented algorithm to create Low/High sample keywords:

By using the above method, we can guarantee that a query containing both keywords will return at least one match. As can be seen, we are not interested in any keyword shorter than 3 letters since a 1 or 2-letter long word is most likely a preposition (e.g. a, on, of, by, ...). Also, we exclude the words “the” and “and” from the list of candidates for a High-Frequency words although they are indeed High-frequency. These kind of High-Frequency keywords are called stop-words and are usually treated differently from other less obvious High-Frequency words.

The reason for choosing a High-frequency word by first examining the second to last word in the title is that it is more likely for a word in that position in a sentence to be a High-Frequency word. This argument is of course based on
some manual observation and is certainly not a rule. If the attempt to identify a
High-Frequency word in that position fails, the algorithm will keep checking the
words in other positions towards the beginning of the expression.

1. Select a Low-Frequency keyword randomly from the meta-database.
2. Form a Query that seeks records containing the selected keyword.
3. Run the Query against citation database.
4. Select a record randomly from the result file.
5. Extract the second to last word in the Title field.
6. If the word is longer than 2 letters and it is not the word "The" and it is not the
   word "And" and it is not the Low keyword selected at the beginning then:
      Select it as a High-Frequency keyword.
   Else,
      Check the third to last word in the title. (If not successful again, keep
      moving towards the beginning of title field and check the words.)
7. Pair the two Low and High Keywords together and save the pair in the output
   file.

Figure 4: The pseudo-code for the Low/High sample-creating algorithm

5.3.1.1 Experiment Results

We ran the experiments using 11,000 sample pairs and stored the results in an
output file. Each line of the output file contained the following items:

- The keyword pair
- Non-Optimized time
- Optimized time
Number of years for each keyword separately (The number for the High-Frequency keyword is 26)

The number of years for the ANDed year constraint.

The following table presents the results of this experiment.

<table>
<thead>
<tr>
<th># of ANDed Year</th>
<th>Number of occurrences</th>
<th>Optimized Time (sec)</th>
<th>Non-Optimized Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>383</td>
<td>.0078</td>
<td>.1460</td>
</tr>
<tr>
<td>1</td>
<td>3730</td>
<td>.2028</td>
<td>.2800</td>
</tr>
<tr>
<td>2</td>
<td>1438</td>
<td>.2398</td>
<td>.2817</td>
</tr>
<tr>
<td>3</td>
<td>897</td>
<td>.2763</td>
<td>.3220</td>
</tr>
<tr>
<td>4</td>
<td>699</td>
<td>.3097</td>
<td>.3583</td>
</tr>
<tr>
<td>5</td>
<td>492</td>
<td>.3551</td>
<td>.3702</td>
</tr>
<tr>
<td>6</td>
<td>454</td>
<td>.3790</td>
<td>.388</td>
</tr>
<tr>
<td>7</td>
<td>374</td>
<td>.448</td>
<td>.4912</td>
</tr>
<tr>
<td>8</td>
<td>276</td>
<td>.4641</td>
<td>.4720</td>
</tr>
<tr>
<td>9</td>
<td>246</td>
<td>.5402</td>
<td>.5434</td>
</tr>
<tr>
<td>10</td>
<td>221</td>
<td>.5534</td>
<td>.5687</td>
</tr>
<tr>
<td>11</td>
<td>216</td>
<td>.5616</td>
<td>.5760</td>
</tr>
<tr>
<td>12</td>
<td>183</td>
<td>.614</td>
<td>.6109</td>
</tr>
<tr>
<td>13</td>
<td>149</td>
<td>.6181</td>
<td>.5878</td>
</tr>
<tr>
<td>14</td>
<td>134</td>
<td>.6751</td>
<td>.6433</td>
</tr>
<tr>
<td>15</td>
<td>134</td>
<td>.6977</td>
<td>.6599</td>
</tr>
<tr>
<td>16</td>
<td>82</td>
<td>.8240</td>
<td>.8030</td>
</tr>
<tr>
<td>17</td>
<td>125</td>
<td>.7312</td>
<td>.6631</td>
</tr>
<tr>
<td>18</td>
<td>98</td>
<td>.8038</td>
<td>.7746</td>
</tr>
<tr>
<td>19</td>
<td>99</td>
<td>.8026</td>
<td>.7144</td>
</tr>
<tr>
<td>20</td>
<td>76</td>
<td>1.0143</td>
<td>.892</td>
</tr>
<tr>
<td>21</td>
<td>88</td>
<td>.995</td>
<td>.9354</td>
</tr>
<tr>
<td>22</td>
<td>66</td>
<td>1.0000</td>
<td>1.0058</td>
</tr>
<tr>
<td>23</td>
<td>85</td>
<td>1.0314</td>
<td>.9024</td>
</tr>
<tr>
<td>24</td>
<td>77</td>
<td>.935</td>
<td>.8462</td>
</tr>
<tr>
<td>25</td>
<td>64</td>
<td>1.1474</td>
<td>1.0287</td>
</tr>
<tr>
<td>26</td>
<td>114</td>
<td>1.1952</td>
<td>1.0900</td>
</tr>
</tbody>
</table>

Table 4: Execution times for Low-AND-High Keyword pair queries.
Since one of the keywords in the pair is always a High-Frequency one, then we can claim that the ANDed Year constraint is exactly identical to the year field associated with the Low-frequency word in the pair.

The above experiment provided the following summarized results:

<table>
<thead>
<tr>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Optimized time</td>
<td>.354 sec</td>
</tr>
<tr>
<td>Average Non-Optimized time</td>
<td>.392 sec</td>
</tr>
</tbody>
</table>

This technique resulted in an average 9.7% improvement.

For cases with shorter year range, the improvement is even more impressive. For example we have gained an average 27.5% improvement for 1-year cases that happen to occur more frequently than other cases. If we calculate the average performance improvement for the first 6 rows only (0-year to 5-year cases), it will be 23%. For the first 11 rows of the Table 4, the average improvement is 18%. These first 11 rows count for 8989 cases of the total 11,000.

The 0-year case in the above table denotes cases where the ANDed Year constraint contains no Year item. The samples for this case were inserted into the sample set to investigate the performance of the cases where the optimized query removes any need to access the main database at all (section 2.1).
5.3.2 Low-OR-High Pairs

In this part of the experiment, we considered forming the disjunctive pairs of Low/High keywords. We are interested in records that contain either or both of the keywords. There are a couple of issues that make this case not very appealing for SQO purposes.

One issue is that we know from the beginning that we can not expect any improvement in terms of shrinking the year range constraint. Since one of the keywords is High-Frequency and we wish to form an ORed pair, consequently the resulted year range will always be a full range (26 years).

Another issue is that we may (but not always) end up with a result file that contains only records with the High-Frequency keyword in their title. It can happen because in the query, we always set a constraint on the maximum number of matching records to be stored in the result file. There might be thousands of matching records for the High-Frequency keyword and the query processor may run into them first and fill up the result file with them even before encountering any matching record for the Low-Frequency keyword. As far as this research is concerned, having a result file containing only one of the keywords (that can be either Low-Frequency or High-Frequency one) although not considered as an ideal result, is still acceptable.
We decided to conduct this experiment despite all the above arguments that suggest no improvement. It can be a real world case. A user may submit a query containing constraints on a Low and a High-Frequency keyword joined by "OR".

We used the same sample file that was created by using the approach described in section 5.3.1. Note that the sample pairs in that file were relevant. By relevant, we mean those pairs that appear at least in one identical record. Of course the importance of this assumption is not as high as that of section 5.3.1 (Low/High ANDed pairs) because here, we are interested in disjunctive combination of keywords meaning that a record containing even one of them is considered a desirable answer.

5.3.2.1 Experiment Results

We performed the experiment using 11,000 sample pairs and measured the times. Since the combination is disjunctive and one of the keywords is High-Frequency, all the sample cases ended up being full range (26 years).

<table>
<thead>
<tr>
<th># of Ored Years</th>
<th>Number of occurrences</th>
<th>Optimized Time (sec)</th>
<th>Non-Optimized Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>11,000</td>
<td>.7602</td>
<td>.7266</td>
</tr>
</tbody>
</table>

Table 5: Execution times for Low-OR-High Keyword pair queries.
As was expected, the technique doesn't yield any performance improvement that is mainly because of the nature of the technique described at the beginning of the section.

However, we can still learn something interesting from the above result. We can look at the difference between the two timings in the above table, as the pure overhead time that includes the overhead caused by executing more lines of code in the program and more importantly the overhead associated with accessing the meta-database. It can be seen that the difference is about 34 msec. Our impression is that considering the size of the meta-database and the extra overhead inside the source code, this estimated overhead time is quite reasonable.

5.3.3 Low-AND-Low Pairs

In this section, we were interested in pairs that contain two Low-Frequency keywords joined by “AND”. The first step was to create a collection of such pairs and use it as the input file to the program that performs the experiment. Since we knew that the title keyword meta-database contains Low-Frequency keywords, it was the best place to pick the keywords for the experiment.

The sample pairs were created by accessing the meta-database for Title keywords and randomly selecting the keywords. It is important to note that we
are not concerned about whether the keywords that contain in a pair are relevant or not. It means that there is no guarantee that submitting a query containing an ANDed constraint on the keywords would return a match.

5.3.3.1 Experiment Results

We performed this technique using a sample file of 11,000 sample keyword pairs. The output file contained the following items:

- The keyword pair
- Non-Optimized time
- Optimized time
- The number of years for each keyword separately.
- The number of years appeared in the ANDed year constraint.

One issue that is challenging is how the summarized data should be presented. We can not use the same kind of tables that were used in Sections 5.2.1 and 5.2.2. At first, it appears to be a good idea to show the timings separately based on the number of years in the final ANDed Year constraint. This criterion has the disadvantage of hiding some aspects of the experiments from the reader and also might be too general. The following example explains the situation in more detail.
Suppose the Year constraint associated with one of the keywords is (1990, 1991, 1992, 1998, 1999) and the one for the second keyword in the pair is (1980, 1982, 1999). The ANDed Year constraint for the above two Year items would be (1999). Then if we had intended to present the result of the experiment based on the number of years in the final ANDed Year constraint, it would have appeared under "1" (there is only one year in the final Year constraint). Now consider the following case:


The above two examples would appear under the same column when summarizing the results and this can be ambiguous and misleading. In the first example the Year constraints have 5 and 3 Year items. The second example has two constraints with 11 and 9 Year items. But the resulted "ANDed" constraint for both cases has only one year item.

We clearly noticed that the above two cases should be presented in a more informative way and should not be appeared under the same column. We should distinguish between cases where two long Year constraints produce an ANDed result that happens to be the same size as the one produced by ANDing two
shorter Year constraints. The optimized timing for these two different cases might be very close but the Non-Optimized (regular) timings are definitely different and we would like to demonstrate that in our presentations.

Because of that, we decided to include the information regarding each individual constraint involved in the samples in our summarized table.

Table 6 shows the results of our experiment for this technique using 11,000 sample pairs. We put the results in groups based on the number of Year items appeared in each constraint regardless of the number of Year items in the final "ANDed" constraint. For example, the before-mentioned case with 5 and 3 items in each constraint, would appear in the paired cell located at the row associated with [5 – 9] and under the column associated with [1 – 4].

Consider the cells containing numbers .4391 and .5470, for example. They depict respectively, the optimized and non-optimized times for all the cases in which the number of Year items in one of the Year constraints is between 20 and 26 and the number of Year items in the other Year constraint is between 5 and 9.

We refrained from narrowing down the table to each individual number of items because that could have caused a very large and hard-to-interpret table.
We performed the experiment for this technique using 11,000 sample pairs. The results are presented in Table 6. As can be seen in the table, the improvement for cases where the Year constraints are not long is quite significant. In fact, these are the dominant cases that affect the overall performance of the technique because they occur more frequently than those cases with very long Year constraint.

<table>
<thead>
<tr>
<th>Number of year constraints</th>
<th>[1 - 4]</th>
<th>[5 - 9]</th>
<th>[10 - 14]</th>
<th>[15 - 19]</th>
<th>[20 - 26]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 - 4]</td>
<td>.0350</td>
<td>.0933</td>
<td>.1446</td>
<td>.1796</td>
<td>.2236</td>
</tr>
<tr>
<td></td>
<td>.2397</td>
<td>.3014</td>
<td>.3742</td>
<td>.3895</td>
<td>.4868</td>
</tr>
<tr>
<td>[5 - 9]</td>
<td></td>
<td>.2406</td>
<td>.3053</td>
<td>.3942</td>
<td>.4391</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.3408</td>
<td>.4285</td>
<td>.4913</td>
<td>.5470</td>
</tr>
<tr>
<td>[10 - 14]</td>
<td></td>
<td></td>
<td>.4517</td>
<td>.5966</td>
<td>.7113</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.4869</td>
<td>.6198</td>
<td>.6656</td>
</tr>
<tr>
<td>[15 - 19]</td>
<td></td>
<td></td>
<td></td>
<td>.7680</td>
<td>.8954</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.6865</td>
<td>1.118</td>
</tr>
<tr>
<td>[20 - 26]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.9229</td>
</tr>
</tbody>
</table>

Table 6: Execution times for Low-AND-Low Keyword pair queries. (sec)

In each paired cell, the upper cell denotes the Optimized time and the lower cell denotes the Non-Optimized time.
The above experiment provided the following summarized results:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average Optimized time:</strong></td>
<td>0.133 sec</td>
</tr>
<tr>
<td><strong>Average Non-Optimized time:</strong></td>
<td>0.327 sec</td>
</tr>
</tbody>
</table>

This technique resulted in an average 60% improvement.

### 5.3.4 Low-OR-Low Pairs

In this technique, pairs are built by two Low-Frequency Title keywords disjunctively (OR) joined. We are interested in records that contain either or both of the keywords in their Title field. Our speculation was that since more matching records could be found for this case (comparing to the case in Section 5.3.3), the execution time for queries would be significantly higher than that of the previous techniques. The main reason for this is that when we create “ORed” pairs the resulting Year constraint tends to be a large rather than short. In other words, the “ORed” Year constraint is, at least, as long as the longest of the Year constraints that form the pair.

#### 5.3.4.1 Experiment Results

We performed the experiment for this technique using 11,000 sample pairs randomly selected. However, the randomly selected keywords are identical to those used in Low-AND-Low technique (section 5.3.3). Table 7 shows the results of our experiment. The interpretation of the rows and columns of Table 7 is exactly the same as described in Section 5.3.3.1 for Table 6.
Table 7: Execution times for Low-OR-Low Keyword pair queries, (sec)

In each paired cell, the upper cell denotes the Optimized time and the lower cell denotes the Non-Optimized time.

The above experiment resulted in the following summarized results:

Average Optimized time: 1.369 sec
Average Non-Optimized time: 2.006 sec

This technique resulted in an average 32% improvement.

This technique also behaved as expected meaning it leads to considerable improvement. However, it’s improvement is not as dramatic as that of Low-AND-Low technique described in 5.3.3. This difference in the amount of gained improvement is not unreasonable considering the very basic difference between the two techniques which is how the keywords have been paired up. As
described earlier, the "ORed" result of two year constraints associated with the two keywords in the pair, tends to be of larger size than both year constraints. On the contrary, the "ANDed" result of two year constraints is expected to be shorter than both Year constraints.

Of course the rule of thumb is, the longer the Year constraint in a query, the longer the query takes to execute. Since our point of reference (non-optimized case) is a full range year constraint (26 years) and we aim to shrink the Year constraint as much as we can, obviously the technique that yields shorter final Year constraint, (here the conjunctive technique) tends to enjoy larger improvement.
6 SQO Techniques for Author Search

6.1 Introduction

This chapter covers two techniques for Author search operations. In the first technique, the submitted query contains only one constraint on the attribute "Author". The second technique, deals with queries that contain one constraint on attribute "Author" and another constraint on attribute "Title" keyword joined by "AND". None of the classes of queries considered in these two techniques originally contain any constraint on attribute "Year".

6.2 Author/Year Technique

Suppose that a query has been submitted that has only one constraint on the attribute Author. This basically means that the user wants to retrieve all the information about a person's publications regardless of the year of publication. As described before, the original query processor searches the entire range of years to find the answer to queries that do not specify any Year constraint. Searching the entire range of years, in this system, means accessing one subdirectory for each year. Since the information stored in the citation database
spans over a period of 26 years (1974 to 1999), the query processor examines 26 subdirectories to prepare the answer.

It is obviously not efficient to access the entire year range, since we know that it is unlikely that an author has publications in all 26 years in the range. So it prompts us that adding a constraint on attribute Year to our original query will keep the query processor from accessing all subdirectories associated with each year. This, in essence, corresponds to the technique called constraint introduction described in section 2.1. In order to eliminate unnecessary access to year files, we need to know in which years the author has publications. This is where the meta-database plays an important role. The query optimizer, first accesses the meta-database to retrieve the rule corresponding to the specified author. Our assumption is that there is always one entry for each author in the meta-database unless the author has publications in all the 26-year period. In other words, no worthy rule is left out of the meta-database. This makes our optimization system complete.

Each record in the meta-database can be considered as a rule with a constraint on Author in its antecedent and a constraint on year in its consequent. The following is an example of such a record (rule):

<table>
<thead>
<tr>
<th>Author</th>
<th>Years</th>
</tr>
</thead>
</table>

Table 8: A sample tuple in the Author-Year Relation of the meta-database
Table 8 specifies the years in which an author called “King” has publications that are listed in the citation database. In section 6.2.1, we explain how the meta-database for Author/Year technique has been built.

Given a query with the constraint:

Select ... Where Author = “King” AND Year = 1999

the system will return null without even accessing the citation database, if it is allowed to go through the above relation in the meta-database.

Consider another query with a constraint like:

Select ... Where Author = “King”

The non-optimized query processor would implicitly add constraints spanning all years from 1974 to 1999. Our system can add “Year” constraints to this query as follows, to form an equivalent query that accesses only the files corresponding to years 1981, 1985, 1991 and 1993:

Select ... Where Author = “King” AND

(Year =1981 OR Year= 1985 OR Year=1991 OR Year=1993).
Even without any accurate measurement, one can conclude that the optimized query executes faster than the non-optimized one, provided that the query does not return hits for all years represented in the database.

### 6.2.1 Building the Meta-database

In this section, the process of building the meta-database for Author/Year technique is described. The final output of this process is a list of Author names along with the corresponding years in which those authors had citations.

Since the size of the citation database is too large to handle in one step, we decided to build the output file in separate parts based on the first letter of the Author's last name and then merge all parts to create the final complete output file. In other words, we first submitted a query asking for records with authors whose names start with letter "A" for the entire year range (26 years). We then, processed the list and created a partial sorted file of authors and years. The same operations were carried out for all the letters of alphabet. The query for retrieving a list of all authors whose names start with a specific letter (e.g. "A") is like the following:
Select Author, Year From citation_database Into Author_A

Where Author = "A*" AND
(Year = 1974 OR .... OR Year = 1999)

The file Author_A is not sorted meaning that records for an identical author but in different years are scattered over the file. We sorted Author_A and combined the records for each author together to make one record that holds the author name and all corresponding years in one place. Figure 5 demonstrates this concept with an example.

After all the partial output files were created, they were all put in one file and an index was built on that file. This concluded the process of building a meta-database for Author/Year technique. The total number of records in the final rule table is close to 6,000,000.

Figure 5: How the Author/Year rules are formed
6.2.2 Experiment Results

Based on the above technique, we performed experiments using a large number of queries with the Author specified. 11,000 specific authors were randomly selected from the meta-database to form author queries. Meta-database for Author-Year rules was created by extracting all authors and their years of publications from the citation database. Table 9 presents the results of this experiment.

<table>
<thead>
<tr>
<th>Years</th>
<th>Number of Cases</th>
<th>Optimized (sec)</th>
<th>Non-Optimized (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5085</td>
<td>.1362</td>
<td>.2197</td>
</tr>
<tr>
<td>2</td>
<td>1969</td>
<td>.1730</td>
<td>.2946</td>
</tr>
<tr>
<td>3</td>
<td>1071</td>
<td>.2171</td>
<td>.4062</td>
</tr>
<tr>
<td>4</td>
<td>1026</td>
<td>.4134</td>
<td>.9315</td>
</tr>
<tr>
<td>5</td>
<td>324</td>
<td>.2795</td>
<td>.3513</td>
</tr>
<tr>
<td>6</td>
<td>247</td>
<td>.3140</td>
<td>.4083</td>
</tr>
<tr>
<td>7</td>
<td>202</td>
<td>.3514</td>
<td>.4567</td>
</tr>
<tr>
<td>8</td>
<td>162</td>
<td>.4097</td>
<td>.5356</td>
</tr>
<tr>
<td>9</td>
<td>140</td>
<td>.4740</td>
<td>.6404</td>
</tr>
<tr>
<td>10</td>
<td>93</td>
<td>.5092</td>
<td>.6290</td>
</tr>
<tr>
<td>11</td>
<td>85</td>
<td>.5684</td>
<td>.7071</td>
</tr>
<tr>
<td>12</td>
<td>91</td>
<td>.6295</td>
<td>.7863</td>
</tr>
<tr>
<td>13</td>
<td>74</td>
<td>.6855</td>
<td>.8656</td>
</tr>
<tr>
<td>14</td>
<td>70</td>
<td>.8403</td>
<td>1.027</td>
</tr>
<tr>
<td>15</td>
<td>62</td>
<td>.9263</td>
<td>1.157</td>
</tr>
<tr>
<td>16</td>
<td>55</td>
<td>.9768</td>
<td>1.309</td>
</tr>
<tr>
<td>17</td>
<td>51</td>
<td>1.126</td>
<td>1.374</td>
</tr>
<tr>
<td>18</td>
<td>28</td>
<td>1.177</td>
<td>1.433</td>
</tr>
<tr>
<td>19</td>
<td>41</td>
<td>1.682</td>
<td>1.869</td>
</tr>
<tr>
<td>20</td>
<td>44</td>
<td>1.971</td>
<td>2.343</td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>1.918</td>
<td>2.169</td>
</tr>
<tr>
<td>22</td>
<td>31</td>
<td>2.693</td>
<td>3.003</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>2.991</td>
<td>3.087</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>3.695</td>
<td>3.432</td>
</tr>
</tbody>
</table>

Table 9: Execution times for Author queries.
The numbers in the first column (Years) of Table 9 denote the number of years that an author has publications. It is important to note that the precision of the function used to measure the time is 1 \( \mu \)sec. The numbers appearing in columns Optimized and Non-Optimized are simply the average of all measurements for their specific year case.

Another important matter is that the 11,000 cases are not distributed equally among the 26-year range. It is logical because statistically speaking; there are very few authors who have publications over a wide range of years. The further we move down the table, the smaller the number of cases for each year becomes. For example, 5085 cases of the total 11,000 were authors with publication(s) in only 1 year. It can also be seen that for 25-year and 26-year cases there are no data in the table meaning that none of the 11,000 sample authors has publications over a 25 or 26 years period.

The above statement prompts a very important point that in order to calculate the overall improvement gained by the above technique, we have to assign weights to each year case. By using a weighted approach, we calculated the following numbers as the overall average times:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Optimized:</td>
<td>.263 sec</td>
</tr>
<tr>
<td>Average Non-Optimized:</td>
<td>.416 sec</td>
</tr>
</tbody>
</table>
We can conclude that our technique has reached an average improvement of about 37% for author queries.

If we consider the first few rows of the table (those cases that occur more frequently) and calculate the average improvement, the improvement will become even better. For example, the average improvement for samples falling between 1-year and 5-year points is 44%. For the first 10 rows in Table 9, the average improvement is 42%. These first 10 rows count for 10,319 cases of the total 11,000 samples.

6.3 Author-Title/Year Technique

In section 5.3, we dealt mainly with pairs of identical items namely Title keywords. In this section, we focus on pairs of a Title keyword and an Author name. However, we only consider the conjunctive join of Title keyword and Author name and we also choose Low-Frequency Title keywords to appear in pairs.

Throughout this project, our objective always was to investigate cases that resemble the real-world circumstances and also promise good improvement. The above assumptions regarding the specification of pairs used in this technique have been taken with this objective in mind.
We will mention briefly other alternatives for this technique in Chapter 9 when we discuss future work.

In this experiment, like others, the first step was to generate a set of samples to be used in the queries. We needed to have relevant pairs of Title keyword and Author name. It means that we needed to guarantee that submitting a query with a conjunctive constraint on the two attributes would return at least one match. Because of this restriction, the method of random selection used in some of the previous experiments was not significant here.

There were two approaches to creating the sample pairs. One was to begin with a set of sample Author names and inspect their matching records and choosing any possible Low-Frequency keyword found in them. This approach proved to be inefficient for two reasons. One reason was that we didn't know from the beginning whether the record(s) that matches the Author name does in fact include any Low-Frequency Title keyword. Since the Author names are chosen randomly, any attempt to find relevant Low-Frequency Title keyword may fail (at least for some of the selected Author names). The most serious flaw of this approach is that checking the keywords in the Title to see if they are Low or High Frequency requires at least one scan of the meta-database that stores the rules for Low-Frequency keywords. (we have mentioned in previous chapters that here, by rule, we mean a record that stores the keyword along with its corresponding Year range, as is shown in Table 1)
There is a second approach that is more efficient than the first one. In this approach, we start by selecting a set of Low-Frequency Title keywords. Remember that we have access to meta-databases of Low-Frequency keywords that can be used for this purpose although originally, they are supposed to be used for rule retrieval in semantic optimization process. Now, the only thing that is remained is to select the related author (or in case of more than one author, one of them). As can be seen, the successful attempt to create the pairs is guaranteed and no additional meta-database scanning is necessary.

6.3.1 Experiment Results

Using the second approach described in the previous section, we generated 22,000 sample pairs of Author names and Low-Frequency Title keywords. To investigate how the optimization works in case of irrelevant pairs, we decided to include a number of them (around 2900) in the sample set. Table 10 shows the results presented based on the same description as Sections 5.2.4.1 and 5.2.3.1.

The summarized results are as follows:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Optimized time:</td>
<td>.1734 sec</td>
</tr>
<tr>
<td>Average Non-Optimized</td>
<td>.3574 sec</td>
</tr>
</tbody>
</table>

This technique resulted in an average 51% improvement.
The interesting matter regarding this technique is that we don't experience any early cross-over point. It means that the technique continues to perform efficiently as the Year range for the items in the pair grows. As can be seen in the Table, even when the Year ranges (constraints) grow as large as [20 – 26] X [10 – 14] the technique still yields improvement. (.6711 seconds Optimized time vs. .7221 seconds Non-Optimized time)

<table>
<thead>
<tr>
<th>[1 – 4]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1396</td>
<td>0.3593</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[5 – 9]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1661</td>
<td>0.2142</td>
</tr>
<tr>
<td>0.3392</td>
<td>0.3404</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[10 – 14]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1577</td>
<td>0.2476</td>
</tr>
<tr>
<td>0.2606</td>
<td>0.3400</td>
</tr>
<tr>
<td>0.3591</td>
<td>0.3266</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[15 – 19]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1747</td>
<td>0.2814</td>
</tr>
<tr>
<td>0.4368</td>
<td>0.4343</td>
</tr>
<tr>
<td>0.3509</td>
<td>0.3566</td>
</tr>
<tr>
<td>0.4699</td>
<td>0.4362</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[20 – 26]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1909</td>
<td>0.3673</td>
</tr>
<tr>
<td>0.6711</td>
<td>0.9640</td>
</tr>
<tr>
<td>0.3653</td>
<td>0.7221</td>
</tr>
<tr>
<td>0.9451</td>
<td>1.305</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of year items in the constraint</th>
<th>[1 – 4]</th>
<th>[5 – 9]</th>
<th>[10 – 14]</th>
<th>[15 – 19]</th>
<th>[20 – 26]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 – 4]</td>
<td>0.1396</td>
<td>0.1661</td>
<td>0.1577</td>
<td>0.1747</td>
<td>0.1909</td>
</tr>
<tr>
<td>[5 – 9]</td>
<td>0.3593</td>
<td>0.2142</td>
<td>0.2476</td>
<td>0.2814</td>
<td>0.3673</td>
</tr>
<tr>
<td>[10 – 14]</td>
<td>0.3404</td>
<td>0.3392</td>
<td>0.2606</td>
<td>0.4368</td>
<td>0.6711</td>
</tr>
<tr>
<td>[15 – 19]</td>
<td>0.3591</td>
<td>0.3392</td>
<td>0.3591</td>
<td>0.4699</td>
<td>0.9640</td>
</tr>
<tr>
<td>[20 – 26]</td>
<td>0.3266</td>
<td>0.3400</td>
<td>0.3266</td>
<td>0.4362</td>
<td>1.305</td>
</tr>
</tbody>
</table>

Table 10: Execution times for Author-Title keyword pair queries. (sec)

In each paired cell, the upper cell denotes the Optimized time and the lower cell denotes the Non-Optimized time.
7 Update Issues

7.1 Introduction

Over time, new records are added to the database or parts of the information already stored are modified. In our system, this means new publications come along information about which should be stored in the citation database. The modification of already stored records is something that might happen very rarely in this system.

It is important to maintain consistency and accuracy between the citation database and the meta-database. Because of the great importance of update issue, it is considered one of the main tasks of database administrators (DBA). The DBA should design a precise update plan that minimizes the odds of missing recent data or experiencing inconsistency. For some time-critical systems, such as stock market information, the update period can be as frequent as 1 minute. For our system (citation database) an update period of one week seems to be adequate.

In addition to the above reasons, there are some important observations regarding database updates and semantic query optimization techniques that are discussed in this thesis. Adding more records to the main database would require
updating the meta-databases used by the SQO module. We explain the above statement by some examples:

Suppose M. Davoodi has published papers in 1989, 1990, 1992 and 1998. The corresponding tuple in the Author/Year meta-database would be as the following:

<table>
<thead>
<tr>
<th>Author name</th>
<th>Year</th>
</tr>
</thead>
</table>

Now, if a new paper published by this author in year 2000 appears in the literature, the corresponding tuple in the meta-database relation should be modified. Otherwise, a query searching for all Davoodi's publications filtered by the SQO module will miss this publication in year 2000. This is an example of updating the meta-database to prevent any inaccuracy processing queries. As stated in Chapter 1, we are not interested in achieving improvement on the expense of losing accuracy. In this example, an addition to the main database dictates a modification in the meta-database.

As another example, consider a case where an author has published a paper for the first time. There is no record on this author in the database. This requires adding a new tuple (rule) to the Author/Year relation of the meta-database. In this case, an addition to the main database dictates an addition to the meta-database.
If there happens to be any keyword in the Title field that can fall within the category of Low-Frequency keywords, it should also be identified and a new tuple for it should be created in the Title-Keyword meta-database. This requires inspecting all the words that appear in the Title field.

In our application, a keyword, once identified as Low-Frequency, will remain Low-Frequency for ever. The reason for this is that all the records that are added to the database are for the current year and not for previous years. So if, for example, a keyword has appeared in all years starting from 1974 to 1999 except 1990 then it is not probable to receive a new record with the keyword in its Title and with 1990 as its year.

Addressing the update issue for meta-databases in our application could be quite straightforward. The reasons for this are twofold. First, we know that changes made to the database are in the form of addition (introducing new tuples). This relieves us from dealing with issues related to tuple modifications. Also it is very unlikely that a record is eliminated from the database. Secondly, update operations can be conducted offline. This is a very advantageous aspect of update operations. Updates can become very resource exhausting tasks so it is essential to make sure that they do not degrade the routine operations of the system. If updates are handled offline, performance degradation can be reduced. Of course, the definition of offline operation may vary from system to system but for our specific system, it has a clear definition as follows.
In the citation database, information is entered by human operators and there is no automated modules that receive (collect) information from any kind of data acquisition system online. An offline state can be defined as when there is no operator entering the information or basically when the operators' working hours are finished. During the offline state, the system may still receive queries from users of the system. So at offline state, no data modification happens but data extraction continues.

We can assume that every week around 20,000 new records are introduced to the system. If the update task is performed on a weekly basis and if it takes about 1 second to perform update for each record, it will take around 20,000 seconds or 5.5 hours to finish it. This amount of time is available during off-hours and will not affect the normal operation of the system drastically. Note that assuming 1 second for each update on the meta-database is considered a long time.

The issue that arises at this point, is that from one update operation to the next, the citation database and the meta-database may run out of synch. How can we maintain accuracy during this period. In the following paragraph, we propose our approach.
Suppose that the information available in the meta-database (inference rules) is only updated until 1997 (it is just an assumption and not the reality). We are aware of the fact that the meta-database lacks information for later years, so in order to get around this problem, we decided to intervene in the automatic process of semantic query optimization. It means that we continue to use the meta-database to retrieve all the useful rules for years up to 1997 according to the techniques presented in the previous chapters and at the final stage when the optimizer has prepared the optimized Year constraint, we always append years 1998, 1999 and 2000 to it. We assume that the three appended year items (post-update years) are required to maintain accuracy of the result set. This might not be needed for most of the queries but its advantage is that we reduce the odds of missing matches to almost zero. If there is an answer to the query for any or all of the years whose rules are not yet reflected in the meta-database, by adopting the above method, we will not miss it. However, for some queries, there might be no answer for any of the post-update years meaning that adding the years to the optimized query is unnecessary. In this case, we call the unnecessary post-update year, redundant year. Since we do not know from the beginning which query needs the post-update years and which do not, we should append the post-update years to all the optimized queries.
7.2 Update Experiments

In this section, we consider a scenario where the citation database is up-to-date but the meta-database used for SQO techniques is not (we can call it a virtual scenario since it is not the case with our application). Like always, the main objective is to maintain the accuracy of the operation. That means using SQO techniques should in no way lead to incomplete (partially complete) result sets.

We conducted a set of experiments to evaluate the performance of the update mechanism that we adopted. We investigated the overhead caused by adding post-update Year constraints that might be in fact redundant for a large number of queries.

As an example, suppose that a query has been submitted inquiring about a specific author's papers and the author has had no more publications since 1995. Obviously, adding the Years 1998, 1999 and 2000 to the optimized (transformed) query is redundant. Redundancy can even be partial. Assume that an author has a publication in 1999 but not 1998 and 2000. In this case years 1998 and 2000 are redundant.

Our speculation was that there should be indeed an overhead associated with manual intervention in SQO process and this overhead should not cause the query execution time to exceed that of a non-optimized query.
We chose the Author/Year technique (Chapter 6) as the framework for our experiments in this section. We used the same sample set used in Chapter 6. In reality, the meta-database was updated until the end of 1999 but we virtually assumed that the meta-database was updated until 1997 and the information for year 2000 was also available in the citation database. Regardless of whether or not files for years 1998, 1999 and 2000 had records satisfying a given query, we appended these post-update Year items to the Year constraint and ran the query and measured the execution times. The average execution time achieved was .361 seconds, which was well in between the two numbers we obtained (Section 6.2.2) by running the optimized experiment (.263 seconds) and non-optimized experiment (.416 seconds). Recall that non-optimized queries scan all the year files in the 26-year range, from 1974 to 1999.

We also measured another factor that was very informative in the sense that it gave us a hint regarding the redundancy issue discussed earlier. We counted the number of cases where none of the files corresponding to post-update years contained records that satisfied a given query (completely redundant) as well as cases where some post-update years did contain records that satisfied the query (partially redundant).

It turned out that 989 cases out of the total of 11,000 samples only received one Year item that they did not need. For example, the author had a publication in
1998 and 1999 but not 2000. In 1639 cases, two of the added Year constraints were not necessary and in 8433 cases, there were no need at all for any of the post-update years hence a completely redundant. Note that despite the very large number of cases with no need for any of the post-update years, the average execution time is still below that of non-optimized process and the accuracy of the system is intact.

We were also curious to investigate what would happen if the meta-database is lacking information for a quite large time period (many Year items). We assumed that the meta-databases are only updated until 1991 and ran an experiment whose essence was completely the same as the first experiment but this time 9 post-update years were added to the Year constraint. Our speculation was that the average execution time should fall somewhere between the time for the first experiment (only 3 post-update years), which was .3618 seconds, and the time for non-optimized, which was .416 seconds. It turned out to be true. The average execution time for this experiment was .401 seconds. In this experiment, 3742 cases out of 11,000 did not need any of the 9 post-update years. 3672 did not need 8 of them and there was no case where all the post-update years were needed.
8 Future Work and Summary

8.1 Future Work

This section discusses the areas that can be considered for further investigation of SQO techniques for our application.

8.1.1 Query-Driven Approach

As mentioned in Section 3.3, we implemented the Data-Driven approach to SQO. The other approach suggested by the researchers, the Query-Driven approach, seems to be an efficient approach as well. It could be a proper research directive to investigate the potentials of Query-Driven approach.

Query-Driven is different from Data-Driven approach in many ways. First, it takes time to build up an accurate and comprehensive rule-based (meta-database) in Query-Driven systems. The system has to wait for new submitted queries to arrive and be used for rule derivation.
Secondly, regarding maintaining the accuracy of the meta-databases, Query-Driven approach is less reliable than Data-Driven approach. The reason is obvious. Relying on queries as a source to derive the rules, most likely, will not lead to a full coverage of all existing rules because the queries are usually biased. For example, it is more common to receive queries on papers published in the recent years rather than an old publication. This may cause a rule for an old publication to remain unidentified. It may be argued that it is still acceptable since that old paper will not be the target of any queries at future. This might be true but it still contradicts with maintaining complete accuracy.

8.1.2 Queries with Triplet or Quadruplet constraints

In our research, we considered forming pairs of constraints for the queries. One research directive could be investigating cases where the "where" clause in the query contains more than two constraints (excluding the Year constraint that always appears).

One example is Triplets of Low-Low-Low Title keywords joined conjunctively or disjunctively. Another example is combination of one Author name and two or more Low-Frequency or High-Frequency Title keywords.

This subject does not require adding any meta-database to the system. The existing meta-databases suffice for investigating this subject.
8.1.3 Title-To-Author Technique

Another technique that can be considered for the citation system is to derive the corresponding Author(s) for a Low-Frequency Title keyword, which constitutes the "Where" clause of a query and adding Author field to the query as a new constraint. It might be useful since we know that there exists an index file on Author field and according to SQO principles, it can be beneficial to add constraints on indexed fields. Note that we are not dealing with Year constraint. We use the full range (26 years) Year constraint.

This case requires building a new meta-database that stores the corresponding Author(s) for each Low-Frequency keyword.

We investigated this case manually for only a few samples and the results didn't prompt any significant improvement. One major drawback in this technique is that usually there are more than one Author associated with each Low-Frequency keyword. It is an extremely rare case where the Low-Frequency Title keyword has appeared only in one or two papers hence adding one or more Author constraints to the optimized query. Most of the Low-Frequency Title keywords are bound to a relatively large number of Author attributes and adding all those Author attributes to the query as constraints does not seem to help the cause of SQO.
However, considering a mix of this technique and the technique described in Chapter 7 (Author-Title/year) may result in significant improvement.

8.2 Summary

By reviewing the results of our experiments, we can certainly conclude that adopting SQO techniques for our specific system is beneficial. All of our experiments except one (Section 5.3.2) resulted in considerable average improvement (for example up to 60% in section 5.2.3) by using SQO techniques.

The overhead associated with accessing the meta-databases is almost negligible. In most of the cases the measured access time to the meta-database is less than 10 msec.

We believe that the most valuable aspect of SQO is that it relieves the implementers from dealing with gory details of low-level operations. SQO takes advantage of indexed attribute but does not involve itself in creating and maintaining index files. As it was the case in our research, acquiring a thorough knowledge of the structures of databases in the system is essential but there was no need for understanding the inside operations of the query processor (not the query optimizer).
There are a number of characteristics that make our proposed techniques very efficient and overhead free. We represent rules as relations in a database and use relational DBMS techniques to search for and apply rules. This reduces the overhead to a minimum. Additionally, the data-driven approach that we adopted allows exhaustive search and extraction of rules in the main database. In other words, we claim that if a query has the potential to be optimized, our proposed techniques will optimize it.

The following graphs present a summarized view of the achieved improvements in various techniques.
Figure 7. Title Keyword/Year
(Section 5.1)

Figure 8. Low-AND-High Pair
(Section 5.2.1)
Figure 9. Low-OR-High Pair (Section 5.2.2)

Figure 10. Low-AND-Low Pair (Section 5.2.3)
Figure 11. Low-OR-Low Pair (Section 5.2.4)

Figure 12. Author-Title/Year (Section 6.3)
Bibliography


[SIEGEL] Michael D. Siegel; “Automatic rule derivation for semantic query optimization”; Boston University; 1989