PROJECT REPORT
ON

Query Optimizer in Database Systems

Submitted in partial fulfillment of the Degree of

Bachelor of Technology

JAYPEE UNIVERSITY OF
INFORMATION TECHNOLOGY

MAY 2007

Under the guidance of
Mr. Vipin Arora
Lecturer
Department of CSE/IT

SUBMITTED BY:
Abhay Agarwal (031206).
Shrimi Sharma (031260).
Bhuvan Sethi (031282).

DEPARTMENT OF COMPUTER SCIENCE
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT.

CERTIFICATE

This is to certify that the work entitled, ***"Query Optimization in Database Systems"*** submitted by

Abhay Agarwal	031206
Bhuvan Sethi	031282
Shrimi Sharma	031260

in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science of the Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

.....

Mr. Vipin Arora

Project Supervisor

ACKNOWLEDGEMENT

No project endeavor is a sole exercise. Various individuals in their own capacity at some point or other contributed in successful completion of the project. In acknowledging their guidance, support and assistance, we humbly thank them.

We would like to express our sincere thanks and gratitude to **Mr. Vipin Arora**, who provided us with all the necessary knowledge about this project. His help throughout made this project a success. He was always there to guide us and helped us solve our problems. His suggestions and criticism of our work were invaluable. His attitude and dedication towards the project really motivated us to come this far.

Finally, we thank each other for constant support and encouragement. The group's unobtrusive support and suggestions bolstered our confidence and made this project a success.

Abhay Agarwal	**Bhuvan Sethi**	**Shrimi Sharma**
031206	031282	031260

Table of Contents

PREFACE

The design of the database is one of the most important factors in the performance of the database and with a good database design you also need queries to perform optimally. Everyone wants the performance of their database to be optimal but does not concentrate on designing a query. They just write the query depending on the only major factor 'What I want'. They don't consider that the same thing can be achieved with some alternate queries and in a more efficient manner.

Query optimization is an area where database systems can achieve significant performance gains. Modern database applications demand optimizers with high extensibility and efficiency. Although more than one decade's effort has been contributed to this area, the state of art in optimizer research is still not adequate for fulfilling user demands.

So we, in our project try to change the query given by a user to make it more efficient. The main goal of our project **"Query Optimization in Database Systems"** is to determine how a query must be processed in order to minimize the user response time.

This report explains in detail the design and implementation of the software for our project. We have also implemented a Word corrector to predict and correct miss spelled column names and table names.

List of figures

Chapter 1

INTRODUCTION

MOTIVATION

Imagine yourself standing in front of an exquisite buffet filled with numerous delicacies. Your goal is to try them all out, but you need to decide in what order. And what exchange of tastes will maximize the overall pleasure of your palate?

Although much less pleasurable and subjective, that is the type of problem a query optimizer is called to solve. Given a query, there are many plans that a database management system (DBMS) can follow to process it. All plans are equivalent in terms of their final output but vary in their cost i.e., the amount of time that they need to run. What is the plan that needs the least amount of time? The cost difference between two alternatives can be enormous. So, query optimization is absolutely necessary in a DBMS.

In spite of the fact that query optimization has been a subject of research for more than fifteen years, query optimizers are still among the largest and most complex modules of database systems, making their development and modification difficult and time consuming tasks. The situation is further complicated by the needs of modern database applications, such as Decision Support Systems (DSS), On-Line Analytical Processing (OLAP), large Data Warehouses (DWH) etc. These new application areas demand new database technologies, such as new query languages and new query processing techniques, which are quite different from those in traditional transaction processing applications.

Over the past several years, several generations of commercial and research query optimizers have been developed, making contributions to the extensibility and efficiency of optimizers.

Query optimization is of great importance for the performance of a relational database, especially for the execution of complex SQL statements. A query optimizer determines

the best strategy for performing each query. **The query optimizer chooses, for example, which join techniques to use when joining multiple tables**. These decisions have a tremendous effect on SQL performance, and query optimization is a key technology for every application, from operational systems to data warehouse and analysis systems to content-management systems.

The query optimizer is entirely transparent to the application and the end-user. Because applications may generate very complex SQL queries, query optimizers must be extremely sophisticated and robust to ensure good performance. **Query optimizers transform SQL statements into equivalent but better performing SQL statements**. Query optimizers are typically ***'cost-based' or 'Rule Based'***. In a cost-based optimization strategy, multiple execution plans are generated for a given query, and then an estimated cost is computed for each plan. The query optimizer chooses the plan with the lowest estimated cost. In the rule based strategy, all the heuristics are already laid out and optimization is based on these pre existing rules.

Objective

- ✓ To develop a software that optimizes queries.
- ✓ To develop a software that implements queries on the database.

Introduction to SDLC

SDLC followed by our group encompasses the following phases:

Post Implementation

Describe post implementation and task like to install the created software and maintain information system and the software

Implementation

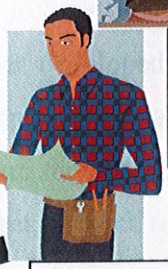
Includes implementation op preparation, implementation on system and resolution of problems identified in development phase

Development

Converts a design into a complete information system. Includes installing system preparing test files, coding, compiling performance test

Design

Transform a detailed requirement into a detailed System Design Focus on how to deliver the required functionality



Requirements Analysis

Analyses user needs and develops user requirement. Creates a detailed functional requirement and document

Planning

Develop a PM plan and other documents. Provides basis for acquiring resources needed to achieve a solution

System Concept Development

Defines the scope and boundary of the concepts. Include Document, Cost benefit Analysis Feasibility, Plan, Study

Initiation

Begins when A user/ sponsor identifies Need or an opportunity. Concept proposal is created

System Development Life Cycle (Eight Phases)

INITIATION PHASE

The initiation of a system (or project) begins in this phase when a need or opportunity for change or completely a new system is identified. A Project Team is appointed to manage the project. This system need is documented in a Concept Proposal. After the Concept Proposal is approved, the System Concept Development Phase begins.

SYSTEM CONCEPT DEVELOPMENT PHASE

Once a System need is approved, the approaches for accomplishing the concept are reviewed for feasibility and appropriateness. The Systems Boundary Document identifies the scope of the system and requires Senior Official approval and funding before beginning the Planning Phase.

PLANNING PHASE

The concept is further developed to describe how the System will operate once the approved system is implemented, and to assess how the system will impact employee and user privacy. To ensure the products and/or services provide the required capability on time and within budget, project resources, activities, schedules, tools, and reviews are defined.

REQUIREMENTS ANALYSIS PHASE

Functional user requirements are formally defined and delineate the requirements in terms of data, system performance, security, and maintainability requirements for the system. All requirements are defined to a level of detail sufficient for the systems design to proceed. All requirements need to be measurable and testable and relate to the System need or opportunity identified in the Initiation Phase.

DESIGN PHASE

The physical characteristic of the system are designed during this phase. The operating environment is established, major subsystems and their inputs and outputs are defined, and the processes are allocated to resources. Everything requiring user input or approval has been documented and reviewed by the user. The physical characteristics of the system are specified and a detailed design is prepared. Subsystems identified during design are used to create a detailed structure of the system. Each subsystem is partitioned into one or more design units or modules. Detailed logic specifications are prepared for each software module.

DEVELOPMENT PHASE

The detailed specifications produced during the design phase are translated into hardware, communications, and executable software. Software shall be unit tested, and retested in a systematic manner. Hardware is assembled and tested.

IMPLEMENTATION PHASE

The system or system modifications are installed and made operational in a production environment. The phase is initiated after the system has been tested and accepted by the user. This phase continues until the system is operating in production in accordance with the defined user requirements.

POST IMPLEMENTATION PHASE

The system operation is ongoing. The system is monitored for continued performance in accordance with user requirements, and needed system modifications are incorporated. The operational system is periodically assessed through In-Process Reviews to determine how the system can be made more efficient and effective. Operations continue as long as the system can be effectively adapted to respond to an organization's needs. When

modifications or changes are identified as necessary, the system may reenter the planning phase.

Chapter 2

INITIATION PHASE

Objective

The Initiation Phase began as the need to develop the software was identified. The objectives of the initiation phase were to:

- ✓ Identify the need to develop a software for Query Optimization.
- ✓ Study databases and relational algebra and validate the need for Query Optimization

Tasks and Activities

This phase involved detailed study of databases as well as relational algebra. A review of the literature is presented below.

Introduction to Databases

A Database is a collection of interrelated data and a Database Management System is a set of programs to use and/or modify this data.

Approaches to Data Management

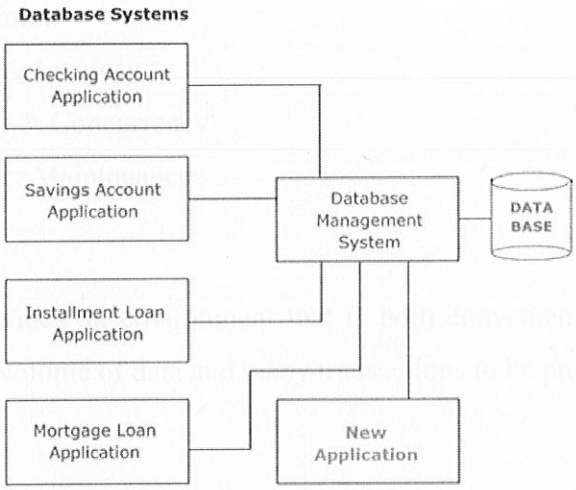
- File-Based Systems
- Database Systems

Drawbacks of File-Based Systems

- Data Redundancy and Inconsistency
- Unanticipated Queries
- Data Isolation
- Concurrent Access Anomalies

- Security Problems
- Integrity Problems

Advantages of Database Systems



As shown in the figure, the DBMS is a central system which provides a common interface between the data and the various front-end programs in the application. It also provides a central location for the whole data in the application to reside.

Due to its centralized nature, the database system can overcome the disadvantages of the file-based system as discussed below.

- Minimal Data Redundancy
- Data Consistency
- Data Integration
- Data Sharing
- Enforcement of Standards
- Application Development Ease
- Better Controls
- Data Independence
- Reduced Maintenance

Functions of a DBMS

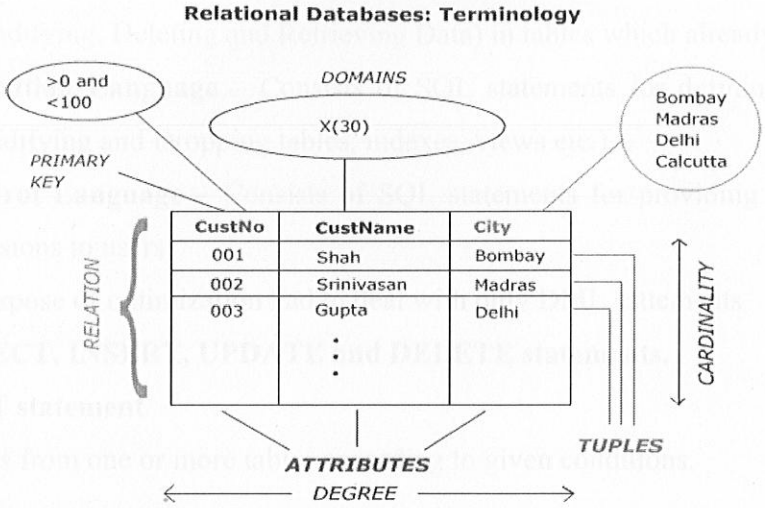
The functions performed by a typical DBMS are the following:

- Data Definition
- Data Manipulation
- Data Security & Integrity
- Data Recovery & Concurrency
- Data Dictionary Maintenance
- Performance

Thus the DBMS provides an environment that is both convenient and efficient to use when there is a large volume of data and many transactions to be processed.

The Relational Model

Relational Databases: Terminology



Properties of Relations

• No Duplicate Tuples – A relation cannot contain two or more tuples which have the same values for all the attributes. i.e., In any relation, every row is unique.

• Tuples are unordered – The order of rows in a relation is immaterial.

• Attributes are unordered – The order of columns in a relation is immaterial.

• Attribute Values are Atomic – Each tuple contains exactly one value for each attribute.

It may be noted that many of the properties of relations follow the fact that the body of a relation is a mathematical set.

Structured Query Language (SQL)

The components of SQL are

a. Data Manipulation Language – Consists of SQL statements for operating on the data (Inserting, Modifying, Deleting and Retrieving Data) in tables which already exist.

b. Data Definition Language – Consists of SQL statements for defining the schema (Creating, Modifying and Dropping tables, indexes, views etc.)

c. Data Control Language – Consists of SQL statements for providing and revoking access permissions to users

We for the purpose of optimization had to deal with only DML sattements

DML – SELECT, INSERT, UPDATE and DELETE statements.

The SELECT statement

Retrieves rows from one or more tables according to given conditions.

General form:

SELECT [ALL | DISTINCT] <attribute (comma)list>

FROM <table (comma)list>

[WHERE <conditional expression>]

[ORDER BY [DESC] <attribute list>

[GROUP BY <attribute (comma)list>]

[HAVING <conditional expression>]

Comparison of different processing strategies

Find all managers who work at a London Branch

```
SELECT *
FROM  Staff s, Branch b
WHERE  s.branchNo = b.branchNo  AND
        (s.position = 'Manager' AND b.city = 'London')
```

For the purpose of this example we assume there are 1000 tuples in staff, 50 tuples in Branch, 50 managers (one for each branch), and 5 London branches. We compare these three queries based on the number of disk accesses required. For simplicity, we assume

that there are no indexes or sort keys on either relation, and that the results of any intermediate operations are stored on disk. The cost of the final write is ignored, as it is the same in each case. We further assume that the tuples are accessed one at a time (although in practice disk accesses would be based on blocks, which would typically contain several tuples), and main memory is large enough to process entire relations for each relational algebra operation.

The first query calculates the Cartesian product of *Staff* and *Branch* which requires (1000 + 50) disk accesses to read the relations, and create the relation with (1000 * 50) tuples. We then have to read each of these tuples again to test them against the selection predicate at a cost of another (1000 * 50) disk accesses, giving a total cost of:

(1000 + 50) + 2*(1000 * 50) = 101050 disk accesses.

The second query joins *Staff* and *Branch* on the branch number *branchNo*, which again requires (1000 + 50) disk accesses to read the relations. We know that the join of the two tuples has 1000 tuples, one for each member of the staff (a member of staff can only work at one branch). Consequently, the Selection operation requires 1000 disk accesses to read the result of the join, giving a total cost of:

2*1000 + (1000 + 50) = 3050 disk accesses.

The final query reads each *Staff* tuple to determine the Manager tuples, which requires 1000 disk accesses and produces a relation with 50 tuples. The second Selection operation reads each branch tuple to determine the London branches, which requires 50 disk accesses and produces a relation with 5 tuples. The final operation is the join of the reduced *Staff* and *Branch* relations, which requires (50 + 5) disk accesses, giving a total cost of:

1000 + 2*50 + 5 + (50 + 5) = 1160 disk accesses.

Clearly the third option is the best in this case, by a factor of 87:1. If we increased the number of tuples in Staff to 10000 and the number of branches to 500, the improvement would be by a factor of 870:1. Intuitively, we may have expected this as the cartasian product and join operations are much more expensive than the Selection operation, and the third option significantly reduces the size of the relations that are being joined together.

Documenting the Phase Effort

A Synopsis of the undertaken project was submitted to Brig(Retd) S P Ghrera, HOD Computer Science and Mr. Vipin Arora, our Project supervisor. After receiving the approval we moved on to the next phase of our project.

Chapter 3

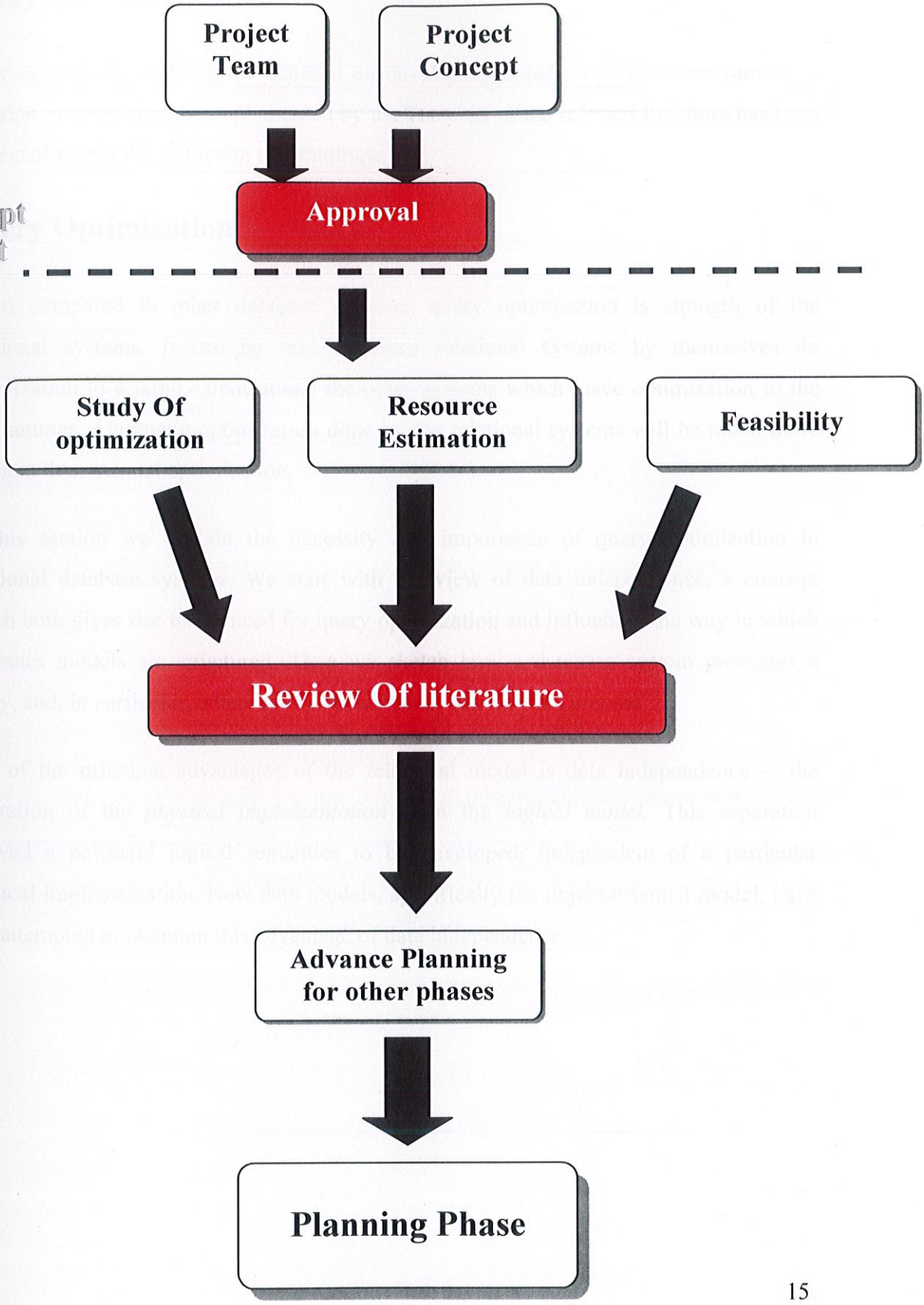
System Concept Development Phase

Objective

The Phase began as the concept proposal was approved in the form of synopsis. The objective of the phase was to submit a report about the basics of query optimization. Thus began the life cycle of an identifiable/tangible project.

Tasks and Activities

This section discusses how we progressed in the System Concept Development phase. A diagrammatic representation has been given in the following page. It depicts the steps followed as we moved along the phase. These steps are described in detail in the later sections.


Project Team
Project Concept
Approval
Study Of optimization
Resource Estimation
Feasibility
Review Of literature
Advance Planning for other phases
Planning Phase

Study and Analysis

After extensively studying the material on developing a query optimizer we came to a decision on what shall be implemented by us. The crux of the relevant literature has been presented within the following subheadings.

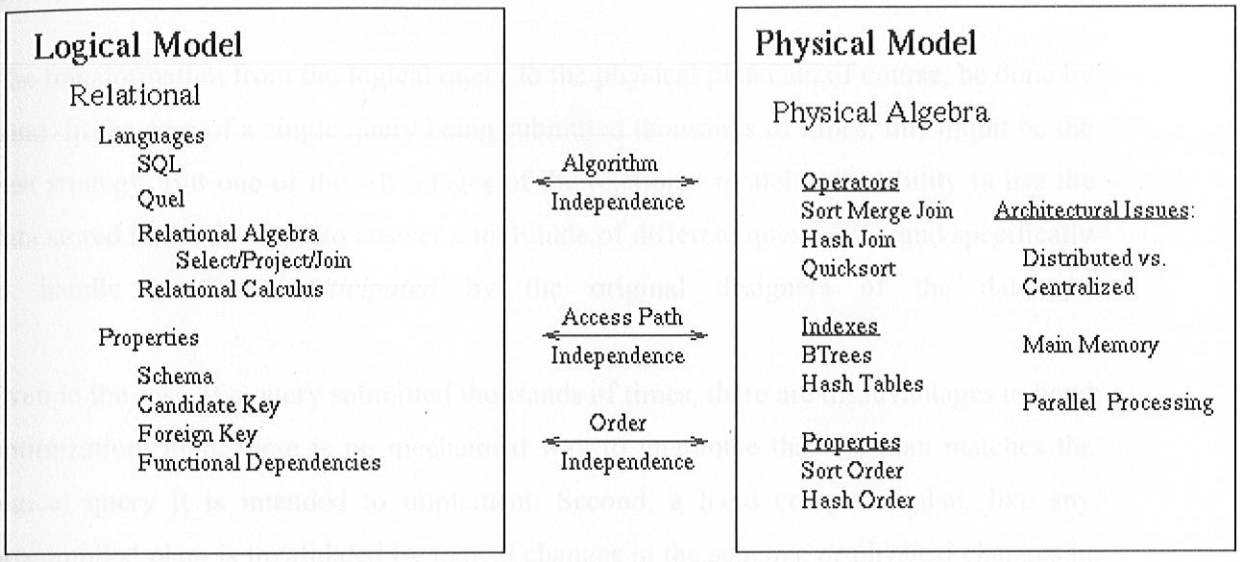
Query Optimization

When compared to other database systems, query optimization is strength of the relational systems. It can be said so since relational systems by themselves do optimization to a large extent unlike the other systems which leave optimization to the programmer. Automatic optimization done by the relational systems will be much more efficient than manual optimization.

In this section we explain the necessity and importance of query optimization in relational database systems. We start with a review of data independence, a concept which both gives rise to the need for query optimization and influences the way in which Cascades models are structured. Then we sketch how a database system processes a query, and, in particular, where query optimization fits into this process.

One of the principal advantages of the relational model is data independence -- the separation of the *physical implementation* from the *logical model.* This separation allowed a powerful logical semantics to be developed, independent of a particular physical implementation. New data models, specifically the object-oriented model, have also attempted to maintain this advantage of data independence.

Data Independence



This figure shows some aspects of data independence in the relational model. In the logical model box (to the left), parts of a logical model are listed including logical query languages and logical properties. In the physical model box (to the right), elements of the physical algebra, access methods, physical properties and elements of the execution engine architecture are listed. The goal of data independence (called physical data independence in Elmasri and Navathe), is that each of the elements shown in the logical model box is independent of all of the elements listed in the physical model box.

One of the challenges of data independence is that database programming becomes a two part process. First, there is the writing of the logical query -- describing *what* the query is supposed to do. Second, there is the writing of the physical plan -- which shows *how* to implement the logical query.

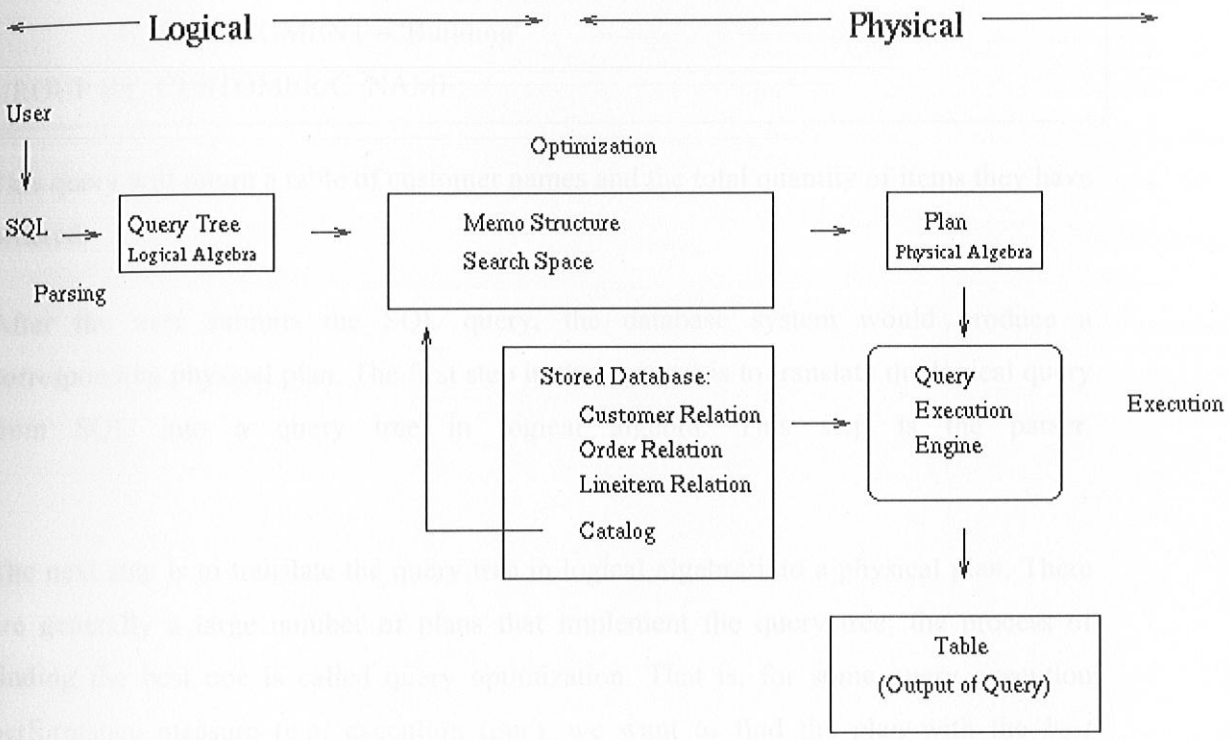
The logical query can be written, in general, in many different forms such as a high level language like SQL, Quel or OQL, or as an algebraic query tree. The physical plan is a query tree in a physical algebra that can be understood by the database system's query execution engine. This physical plan can be thought of as a program that the query execution engine can execute.

The transformation from the logical query to the physical plan can, of course, be done by hand. In the case of a single query being submitted thousands of times, this might be the best strategy. But one of the advantages of the relational model is the ability to use the data stored in the database to answer a multitude of different questions -- and specifically to handle queries *unanticipated* by the original designers of the database.

Even in the case of a query submitted thousands of times, there are disadvantages to hand optimization. First, there is no mechanical way to guarantee that the plan matches the logical query it is intended to implement. Second, a hand computed plan, like any precompiled plan, is invalidated by logical changes in the schema, or physical changes in the access path or physical storage of the data. Third, if query parameters, (or any characteristics of the data in the database) change, the optimality relationship of one plan over another may change. Finally, it requires a database expert, knowledgeable about both the logical database model and the particular physical implementation, to do the hand optimization.

So we look to a mechanical process to translate the logical query, written, for example, in SQL, into a physical plan.

Overview of Query Processing



This figure shows a graphical overview of how a query might be executed in a database system using a Cascades style optimizer. The stored database consists of three relations, Customer, Order and Lineitem, the system catalog and other information required by the database system. The user submits a query written in SQL.

For example, the user might submit the query:

```
SELECT CUSTOMER.C_NAME,  SUM(LINEITEM.L_QUANTITY)
FROM    CUSTOMER,  ORDER,  LINEITEM
WHERE CUSTOMER.C_CUSTKEY =  ORDER.O_CUSTKEY            AND
          ORDER.O_ORDERKEY   =  LINEITEM.L_ORDERKEY      AND
          C_MKTSEGMENT = 'Building'
GROUP BY  CUSTOMER.C_NAME;
```

This query will return a table of customer names and the total quantity of items they have ordered.

After the user submits the SQL query, the database system would produce a corresponding physical plan. The first step in this process is to translate the logical query from SQL into a query tree in logical algebra. This step is the parser.

The next step is to translate the query tree in logical algebra into a physical plan. There are generally a large number of plans that implement the query tree; the process of finding the best one is called query optimization. That is, for some query execution performance measure (e.g. execution time), we want to find the plan with the *best* execution performance. The goal is that the plan be optimal or near optimal within the search space of the optimizer.

The optimizer starts by copying the relational algebra query tree into its search space. The optimizer then expands the search space and finds the best plan.

At this level of generality, the optimizer can be viewed as the code generation part of a query compiler for the SQL language, that produces code to be interpreted by the query execution engine, except that the optimizer's emphasis is on producing "very efficient" code. For example, the optimizer uses the database system's catalog to get information (e.g. number of tuples) about the stored relations referenced by the query, something traditional programming language compilers normally do not do.

Finally, the optimizer copies the optimal physical plan out of its memo structure and sends it to the query execution engine. The query execution engine executes the plan using the relations in the stored database as input, and produces the table of customer names and item quantities as output.

The next section will introduce some of the concepts used in solving the optimization problem.

Fundamental Concepts in Query Optimization

There are many possible ways to express a complex query using SQL. The style of SQL submitted to the database is typically that which is simplest for the end-user to write or for the application to generate. However, these hand-written or machine-generated formulations of queries are not necessarily the most efficient SQL for executing the queries. For example, queries generated by applications often have conditions that are extraneous and can be removed. Or, there may be additional conditions that can be inferred from a query and should be added to the SQL statement. **The purpose of our project is to transform a given SQL statement into a semantically-equivalent SQL statement** (that is, a SQL statement which returns the same results) which can provide better performance. All of these transformations are entirely transparent to the application and end-users; SQL transformations occur automatically during query optimization. SQL transformations implemented by us broadly fall into two categories:

Heuristic query transformations: These transformations are applied to incoming SQL statements whenever possible. These transformations always provide equivalent or better query performance, so that the optimizer knows that applying these transformations will not degrade performance.

Cost-based query transformations: Given a query, there are many logically equivalent algebraic expressions and for each of the expressions, there are many ways to implement them as operators. Even if we ignore the computational complexity of enumerating the space of possibilities, there remains the question of deciding which of the operator trees consumes the least resources. Resources may be CPU time, I/O cost, memory, communication bandwidth, or a combination of these. Therefore, given an operator tree of a query, being able to accurately and efficiently evaluate its cost is of fundamental importance. An optimizer's cost model includes cost functions to evaluate the cost of operators, statistics and formulas to predict the sizes of intermediate results. Two common objectives are minimum total cost and minimum response time. Total cost is the sum of all times incurred in processing the operations of query at various database nodes and in transferring intermediate results among participating database nodes. Response time of query is the time elapsed for executing query. We adopt the objective of minimum total cost in our query optimization with making the best use of parallelism if possible to reduce response time. **Using this approach, the transformed query is compared to the original query, and optimizer will then select the best execution strategy.**

Project Approach

Approach 1: The Optimization Oracle

(Definitely not to be confused with the company of the same name means works for all)

We will like to get the following information, but in 0 time:

- Consider each possible plan in turn.
- Run it & measure performance.
- The one that was fastest is the keeper.

Approach 2: Make Up a Heuristic & See if it Works

- Always use NL-Join (indexed inner whenever possible)
- Order relations from smallest to biggest
- "Syntax-based or Rule-based " optimization

Approach 3: We have three important issues:

- Define plan space to search
- Do cost estimation for plans
- Find an efficient algorithm to search through plan space for "cheapest" plan

We aim at making the Query Optimizer. While trying to formulate an approach to develop our software and going through various research papers and white papers we have adopted Approach 4.

Approach 4: A set of predefined parameters and heuristics was formulated, when a query is given it is analyzed on the basis of the previously defined set and if any of the present rules can be applied on the given input query, it is optimized based on that rule.

Now let us take an example where we want the order id's from the table order where the conditions are that the freight should be greater than 50 or the required date should be less than 5/1/1997. The regular query would be:

```
SELECT O.ORDERID
FROM ORDERS O
WHERE O.FREIGHT>50
UNION
SELECT O.ORDERID
FROM ORDERS O
WHERE O.REQUIREDDATE<5/1/1997
```

Now using the approach 4 the optimized query would be:

```
SELECT O.ORDERID
FROM ORDERS O
WHERE O.FREIGHT > 50 OR O.REQUIREDDATE < '5/1/1997'
```

Feasibility

After the study of all the approaches available to us for the development of the query optimizer we came to a conclusion to use the so called approach 4 as the other approaches were not feasible and we did not have the required resources to follow them.

Documenting the Phase Effort

A report 'Review of Literature' was submitted to our project guide. Concisely the scope and the vision of the project along with the limitations and constraints were also mentioned in the document.

By the study done in this phase we learnt that the term ***optimization*** is actually a misnomer because in some cases the chosen execution plan is not the optimal (best) strategy – it is just a reasonably efficient strategy for executing the query. Finding the optimal strategy is usually too time-consuming except for the simplest of queries and may require information on how the files are implemented and even on the contents of

the files – information that may not be fully available in the DBMS catalog. Hence, planning of an execution strategy may be more accurate description than query optimization.

By the end of this phase we had finalized the approach that would be used for the accomplishment of the project.

Chapter 4

Planning

Objective

This phase was one of the most important phases in the SDLC. The objective of the phase was to come to a decision about the languages and packages to be used to develop the software as well as the interface for the software.

Tasks and Activities

Establishing the Application Environment

The application environment should be conductive to the current system. Lot of team effort was spent on deciding such an environment. A complete phase review activity for all the phases conducted before this phase was conducted thoroughly on the basis of which an application environment was finalized.

Developing Platforms

Databases

Many options were available to us regarding the databases or backend that we could use in the development of the project like Microsoft SQL Server, Oracle, MS access, DB2 etc. We short listed a few and decided to use Microsoft SQL Server. However, our first demo was prepared using Oracle as the backend.

SQL server

Microsoft SQL Server is a relational database management system (RDBMS may be a DBMS in which data is stored in the form of tables and the relationship among the data is also stored in the form of tables). The primary query language is Transact-SQL, an implementation of the ANSI/ISO standard Structured Query Language (SQL) used by both Microsoft and Sybase.

Microsoft SQL Server and Sybase/ASE both communicate over networks using an application-level protocol called Tabular Data Stream (TDS). The TDS protocol has also been implemented by the Free TDS project in order to allow more kinds of client applications to communicate with Microsoft SQL Server and Sybase databases. Microsoft SQL Server also supports Open Database Connectivity (ODBC). SQL Server 2005 also supports the ability to deliver client connectivity via the Web Services SOAP protocol. This allows non-Windows Clients to communicate cross platform with SQL Server. Microsoft SQL Server 2005 also features automated database mirroring, failover clustering, and database snapshots.

Microsoft and other vendors provide a number of software development tools designed to allow business applications to be developed using the data stored by Microsoft SQL Server. Microsoft SQL Server 2005 now includes the common language runtime (CLR) component for Microsoft .NET. Applications developed with .NET languages such as Visual Basic can implement stored procedures and other functions. Older versions of Microsoft development tools typically use APIs to access Microsoft SQL Server functionality. Rapid application development tools incorporate native database gateways for high speed database access and automatic table drill-down for the creation of quick prototype applications for viewing, editing and adding data to any table in the database.

Interface

In order to give a presentable interface to our software, various packages were analyzed. Visual Basic 6 was known to our team but it was not used as it was outdated. After tremendous research we decided upon Visual Basic.NET.

Visual Basic

Visual Basic .NET (VB.NET) is an object-oriented computer language that can be viewed as an evolution of Microsoft's Visual Basic (VB) implemented on the Microsoft .NET framework.

The great majority of VB.NET developers use Visual Studio .NET as their integrated development environment (IDE). SharpDevelop provides an open-source alternative IDE. Like all .NET languages, programs written in VB.NET require the .NET framework to execute.

The original Visual Basic .NET was released alongside Visual C# and ASP.NET in 2002. C# — widely touted as Microsoft's answer to Java — received the lion's share of media attention, while VB.NET (sometimes known as VB7) was not widely covered. As a result, few outside the Visual Basic community paid much attention to it.

Those who did try the first version found a powerful but very different language under the hood.

Documenting the Phase Effort

This phase marked the end of semester 7. A report of the work done was submitted to our project guide. The report consisted of details about query Optimization and the packages that would be used to develop the software. A demo of the interface was also prepared.

Chapter 5

REQUIREMNT ANALYSYS

Objective

This phase consisted of gathering as much knowledge as possible about the requirements and needs of the user.

Tasks and Activities

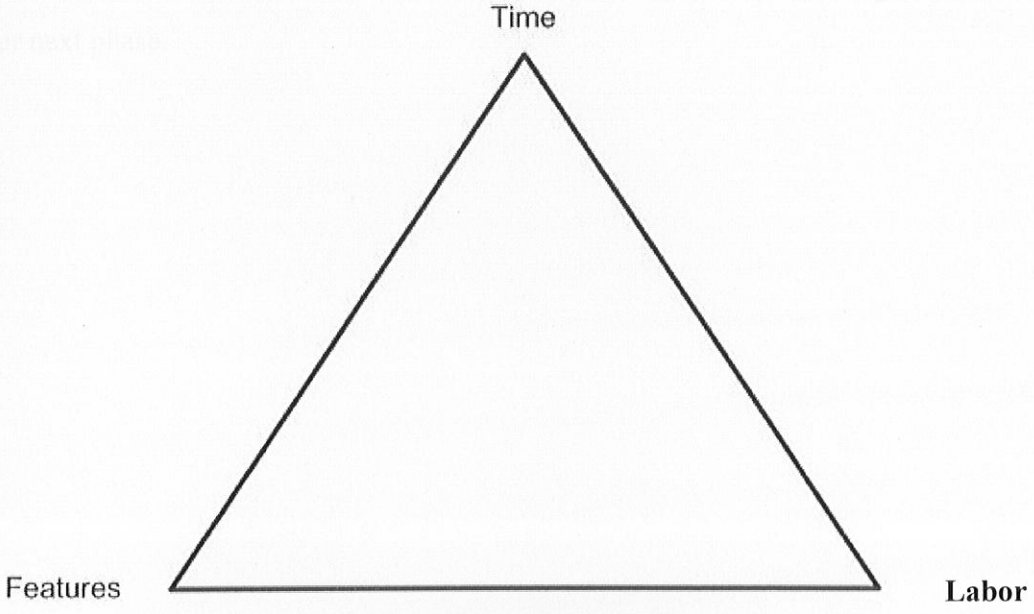
Once we had determined what application we will be developing, it was important to then decide what specifically the application will do. We wanted to define its basic functionality, along with certain features that the software will implement. As we were developing this software for ourselves, this stage in the development process was fairly informal, but it was nevertheless very important. If we had been developing the software for a client or an employer then creating a list of features and requirements would have been vital. The list would have determined what our responsibilities were as a software developer and would have given a clear definition of what would be required for our project to be considered complete. It was essential to remember that clients have a tendency to request additional features as the project progresses while not wanting to pay any additional funds to implement them. In our case, our project guide was playing the role of a client and the cost was most importantly the time and the deadline we had to meet.

In addition to listing features and requirements, we also developed a timeline. We scheduled significant milestones such as project completion and major phases of work being done. For example, we set a milestone for delivery of a working demonstration of the project.

As part of requirement analysis we developed a concept which is used to maintain a balance between the labor, features and time. This concept is discussed in the following section.

Development Triangle

On the subject of labor and features and timelines, it is important to understand the balance that must exist between the three. Any client will undoubtedly want as many features as possible while utilizing minimal resources and within a short time period. It is important to realize that sometimes there is a need to reject a requested feature or timeline in order to keep everything in balance. Imagine that time, features, and labor are three corners of a triangle:



While we can stretch the corners of this triangle, we cannot change the area it occupies, as the area of the triangle represents our total resources. The impact of this is that every section can only increase at the expense of the other two: If we want the project to have more features we will have to either take more time or utilize more labor (additional tools or developers). If we want the project to cost less money(cost of labor), we either need to decrease the number of features or allow the project to take more time (this is because

either you cut back on the number of developers or allow them to only work on the project in their spare time). Finally, if you want the project done faster we either have to decrease the number of features or use more labor or more tools.

A fair balance had to be struck among the three. Our manpower was fixed to three, so we had to determine the limitations on the other two.

Documenting the Phase Efforts

As we were developing this software as part of our final semester project and not for marketing purposes the job of requirement analysis did not prove to be very tedious. The basic requirements were that firstly the software should be easy to use and understand and secondly the query given by the user must be analyzed, optimized and it must output the results also. These issues were addressed and we got down to designing which was our next phase.

Chapter 6

DESIGN

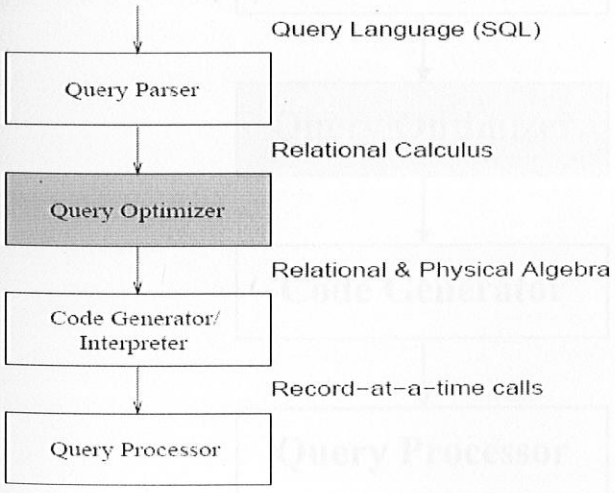
Objective

The objective of the Design Phase was to transform the detailed, defined requirements into complete, detailed specifications for the system to guide the work of the Development Phase. The decision made in this phase address, in detail, how the system will meet the defined functional, physical, and data requirements. Design Phase activities have been conducted in an interactive fashion, producing first a general system design that emphasizes the functional features of the system, then a more detailed system design that expands the general design by providing all the technical details.

Tasks and Activities

Design of the Application

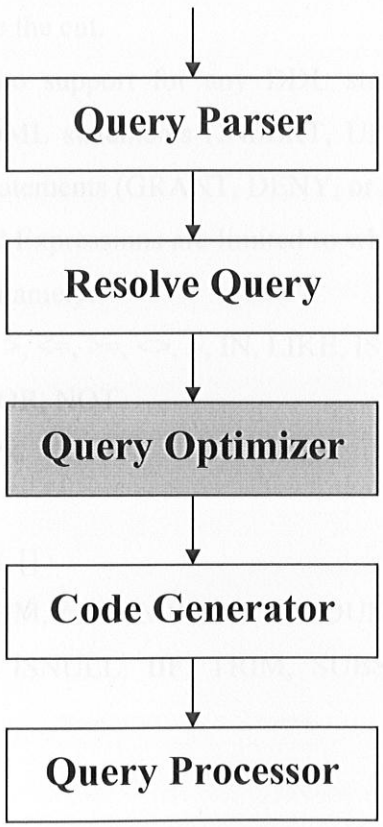
The path that a query traverses through a DBMS until its answer is generated is shown in



The system modules through which it moves have the following functionality:

- ✓ The Query Parser checks the validity of the query.
- ✓ The Query Optimizer examines all algebraic expressions that are equivalent to the given query and chooses the one that is estimated to be the cheapest.
- ✓ The Code Generator or the Interpreter transforms the access plan generated by the optimizer
- ✓ The Query Processor actually executes the query.

We in our application have followed the same approach Queries are posed to our application by interactive users or by programs written in general-purpose programming languages (e.g. VB.NET in our case) that have queries embedded in them. An interactive (ad hoc) query given by the user goes through the entire path as shown in the Figure.



The area of query optimization is very large within the database field. It has been studied in a great variety of contexts and from many different angles, giving rise to several diverse solutions in each case. The purpose of this application is to primarily discuss the core problems in query optimization and their solutions, and only touch upon the wealth of results that exist beyond that.

Module Functionality

Query Parser

This module parses the query .It checks for the validity of the query. This module acts as the syntactical parser it checks the syntax and also has predictive word capabilities. The output of the module is a valid query that if executed will run. The application developed supports only the query that starts with select basically, although we attempted to follow the Microsoft SQL Server 2000 dialect of the SQL language, there were quite a few features that did not make the cut.

Consequently, there is no support for any DDL statements (ALTER, CREATE, or DROP) nor any other DML statements (INSERT, UPDATE, or DELETE) other than SELECT nor any DCL statements (GRANT, DENY, or REVOKE).

Operators, Functions, and Expressions are limited to what is natively supported (or easily translated) by ADO.Net, namely:

Comparison: <, >, <=, >=, <>, =, IN, LIKE, IS NULL

Logical: AND, OR, NOT

Math: +, -, *, /, %

String: +

Wildcards: *, %, []

Aggregation: SUM, AVG, MIN, MAX, COUNT, STDEV, VAR

Functions: LEN, ISNULL, IIF, TRIM, SUBSTRING, CONVERT, LOWER, UPPER

Resolve Query

After the parser has parsed the query and checked for its validity this module is used to resolve the different column alias and table alias This Module is important for proper functioning of the optimizer as it doesn't recognize the alias name and require the table names

There is a checklist of things resolve module do:

- ✓ Resolve any table aliases
- ✓ Check the table names against the DataSet
- ✓ Check the column names against the DataSet
- ✓ Translate column names, operators, etc in expressions
- ✓ Translate a few things to meet the ADO.Net native features

If GROUP BY is used, every item in the select list must be covered by a corresponding item in the GROUP BY list or be a vector aggregate. A quick SQL syntax check is preformed i.e. Expressions in the GROUP BY clause must match the expression in the SELECT clause.

Optimize Query

This is The main Module of the project. It examines the execution plans for query produced in the previous stage and changes it to the cheapest plan to be used to generate the answer of the original query. It employs a search strategy, which examines the space of execution plans in a particular fashion. This module and the search strategy determine the cost, i.e., running time, of the optimizer itself, which should be as low as possible. The SQL query examined by the previous modules is compared based on estimates of their cost so that the cheapest may be chosen. These costs are derived by the heuristics set first based on the theoretical knowledge of the subject, the optimizer.

A major challenge in the design of a query optimizer is to ensure that the set of feasible plans in the search space contains efficient plans without making the set too big to be generated practically. For that purpose, most commercial database systems often have multiple levels of optimization. For example, a system can have a \low" level of

optimization that employs a polynomial-time greedy method or a randomized algorithm, and a \high" level that searches all bushy plans using a conventional dynamic programming enumeration method

We in our application have tried to demonstrate an optimizer and not build open fill running model we have only tried to build a DEMO so we decided to limit of search space to one and deciding the optimized query between the two. The most important thing is the **SQL TRANSFORMATION this can also be refered to as code Generator.**

There are many possible ways to express a complex query using SQL. The style of SQL submitted to the database is typically that which is simplest for the end-user to write or for the application to generate. However, these hand-written or machine-generated formulations of queries are not necessarily the most efficient SQL for executing the queries. For example, queries generated by applications often have conditions that are extraneous and can be removed. Or, there may be additional conditions that can be inferred from a query and should be added to the SQL statement. The purpose of SQL transformations is to transform a given SQL statement into a semantically-equivalent SQL statement (that is, a SQL statement which returns the same results) which can provide better performance. All of these transformations are entirely transparent to the application and end-users; SQL transformations occur automatically during query optimization. We have implemented a wide range of SQL transformations. These broadly fall into two categories:

- ✓ Heuristic query transformations: These transformations are applied to incoming SQL statements whenever possible. These transformations always provide equivalent or better query performance, so that application knows that applying these transformations will not degrade performance.
- ✓ Cost-based query transformations: The application uses a cost-based approach for several classes of query transformations. Using this approach, the transformed query is compared to the original query, and application then selects the best execution strategy. The following sections discuss several examples of

application's transformation technologies. We have tried to implement the heuristic query transformations only.

Heuristic query transformations

Subquery "flattening"

As an example of the techniques in this area, consider the following query which selects those departments that have employees that make more than 10000:

```
SELECT D.DNAME
FROM DEPT D
WHERE D.DEPTNO IN (
                    SELECT E.DEPTNO
                    FROM EMP E
                    WHERE E.SAL > 10000)
```

Without any transformations, the execution plan for this query would be similar to:

OPERATION OBJECT_NAME OPTIONS

SELECT STATEMENT

FILTER

TABLE ACCESS DEPT FULL

TABLE ACCESS EMP FULL

With this execution plan, all of the EMP records satisfying the sub query's conditions will be scanned for every single row in the DEPT table. In general, this is not an efficient execution strategy. However, query transformations can enable much more efficient plans.

One possible plan for this query is to execute the query as a 'semi-join'. A 'semi join' is a special type of join which eliminates duplicate values from the inner table of the join (which is the proper semantics for this sub query). In this example, the optimizer has chosen a hash semi-join

OPERATION OBJECT_NAME OPTIONS

SELECT STATEMENT

HASH JOIN SEMI

TABLE ACCESS DEPT FULL

TABLE ACCESS EMP FULL

Since SQL does not have a direct syntax for semi-joins, this transformed query cannot be expressed using standard SQL. However, the transformed pseudo-SQL would be:

```
SELECT DNAME FROM EMP E, DEPT D
WHERE D.DEPTNO = E.DEPTNO
AND E.SAL > 10000;
```

Sub query flattening is a fundamental optimization for good query performance.

Transitive predicate generation

In some queries, a predicate on one table can be translated into a predicate on another table due to the tables' join relationship. Our application changes the SQL that will deduce new predicates in this way; such predicates are called transitive predicates. For example, consider a query that seeks to find all of the line-items that were shipped on the same day as the order data:

```
SELECT COUNT (DISTINCT O_ORDERKEY)
FROM ORDER, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE = L_SHIPDATE
AND O_ORDERDATE BETWEEN '1-JAN-2002' AND '31-JAN-2002'
```

Using transitivity, the predicate on the ORDER table can also be applied to the LINEITEM table:

```
SELECT COUNT (DISTINCT O_ORDERKEY)
FROM ORDER, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE = L_SHIPDATE
AND O_ORDERDATE BETWEEN '1-JAN-2002' AND '31-JAN-2002'
AND L_SHIPDATE BETWEEN '1-JAN-2002' AND '31-JAN-2002'
```

The existence of new predicates may reduce the amount of data to be joined, or enable the use of additional indexes.

Common sub expression elimination

When the same sub expression or calculation is used multiple times in a query, our application will change the Query such that it will only evaluate the expression a single time for each row.

Consider a query to find all employees in Dallas that are either Vice Presidents or with a salary greater than 100000.

```
SELECT * FROM EMP, DEPT
WHERE
(EMP.DEPTNO = DEPT.DEPTNO AND LOC = 'DALLAS' AND SAL > 100000)
OR
(EMP.DEPTNO = DEPT.DEPTNO AND LOC = 'DALLAS' AND JOB_TITLE =
'VICE PRESIDENT')
```

The optimizer recognizes that the query can be evaluated more efficiently when transformed into:

```
SELECT * FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO AND
LOC = 'DALLAS' AND
(SAL > 100000 OR JOB_TITLE = 'VICE PRESIDENT');
```

With this transformed query, the join predicate and the predicate on LOC only need to be evaluated once for each row of DEPT, instead of twice for each row.

Join Selectivities

The JOIN operation is one of the most time consuming operations in query processing. A join operation matches two tables across domain compatible attributes. One common technique for performing a join is a nested (inner-outer) loop or brute force approach. In this case, for every row in the first table a scan of the second table is performed and every record is tested for satisfying the join condition. A second technique is to use an access structure or index to retrieve the matching records. In this case, for every row in the first table an index is used to access the matching records from the second table.

One factor that significantly affects performance of the join is the percentage of rows in one table that will be joined with rows in the other table. This is called the join selection factor. This factor depends not only on the two tables to be joined, but also on the join fields if there are multiple join conditions between the two tables. For example tests consider a case

```
SELECT *
FROM Patient, Physician
WHERE Patient.SSN = Physician.Dr_SSN
```

This query joins each Physician row with the Patient rows. Each physician is expected to exist once in the Patient table (after all, a physician is also a patient), but 999,000 patient rows will not be joined. Suppose indexes exist on each of the join attributes. There are two options for performing the join.

The first retrieves each Patient row and then uses the index into the Physician table to find the matching record. In this case, no matching records will be found for those patients who are not also physicians.

The second option first retrieves each Physician row and then uses the index into the Patient table to find the matching Patient row. In this case, every physician will have one matching patient row.

It is clear that the second option is more efficient than the first option. This occurs because the join selection factor of Physician with respect to the join condition is 1. Conversely, the Patient selection factor with respect to the same join condition is

1,000/1,000,000. Choosing optimum join methods requires that various table sizes and other statistics be used to compute estimated join selectivities.

```
SELECT *
FROM Patient, Physician
WHERE Physician.Dr_SSN = Patient.SSN
```

If join selectivities are not used, then these two queries can exhibit quite different performance.

Removing DISTINCT

Carefully evaluate whether your SELECT query needs the DISTINCT clause or not. Some users automatically add this clause to every one of their SELECT statements, even when it is not necessary.

The DISTINCT clause should only be used in SELECT statements if you know that duplicate returned rows are a possibility, and that having duplicate rows in the result set would cause problems with your application.

The DISTINCT clause creates a lot of extra work for SQL Server, and reduces the physical resources that other SQL statements have at their disposal. Because of this, only use the DISTINCT clause if it is necessary

For example lets consider a query to select unique oderid from order:

```
SELECT DISTINCT ORDERID FROM ORDERS
```

This query uses a distinct on an primary key of the table Hence in this case DISTINCT shouldn't be used hence the optimized query is

```
SELECT ORDERID FROM ORDERS
```

Remove Query that includes code that doesn't do anything. This may sound obvious, but I have seen this in some off-the-shelf SQL Server-based applications. For example, you may see code like this:

```
SELECT column_name FROM table_name
WHERE 1 = 0
```

When this query is run, no rows will be returned.
Obviously, this is a simple example (and most of the cases where these example can exist have been very long queries), a query like this (or part of a larger query) like this doesn't perform anything useful, and shouldn't be run. It is just wasting SQL Server resources.

By default, many user, especially those who have not worked with SQL Server before, routinely include code similar to this in their WHERE clauses when they make string comparisons:

For example let us consider a query to select **order id** from the table orders where **customer id** is VINET

```
USE NORTHWIND
SELECT ORDERID
FROM ORDERS
WHERE UPPER(CUSTOMERID) = 'VINET'
```

In other words, these developers are making the assuming that the data in SQL Server is case-sensitive, which it generally is not you don't need to use LOWER or UPPER to force the case of text to be equal for a comparison to be performed. Just leave these functions out of your code. This will speed up the performance of your query, as any use of text functions in a WHERE clause hurts performance.

If your database has been configured to be case-sensitive The above example is still poor coding. If you have to deal with ensuring case is consistent for proper comparisons, use the technique described below, along with appropriate indexes on the column in question:

```
USE Northwind
SELECT orderid
 FROM orders
WHERE customerid = 'vinet' OR  customerid = 'VINET'
```

This code will run much faster than the first example.

The application doesn't allow the GROUP BY clause without an aggregate function

The GROUP BY clause can be used with or without an aggregate function. But if you want optimum performance, **don't use the GROUP BY clause without an aggregate function**. This is because you can accomplish the same end result by using the DISTINCT option instead, and it is faster.

For example, if we want to select unique **order id** from the table Order Details where the unit price is greater than 10. We could write the query in two different ways:

```
USE Northwind
SELECT OrderID
FROM [Order Details]
WHERE UnitPrice > 10
GROUP BY OrderID
```

Or

```
USE Northwind
SELECT DISTINCT OrderID
FROM [Order Details]
WHERE UnitPrice > 10
```

Both of the above queries produce the same results, but the second one will use fewer resources and perform faster.

Be careful when using OR in your WHERE clause, it is fairly simple to accidentally retrieve much more data than you need, which hurts performance. For example, take a look at the query below:

If we want to select **order id** from the table orders where either the freight is greater than 50 and employee id is 5 and required date is less than 5/1/1997 or the freight is greater than 50 and customer id is VINET and required date is less than 5/1/1997 or the required date is less than 5/1/1997 or the freight is greater than 50.

```
USE Northwind
SELECT O.ORDERID
FROM ORDERS O
WHERE O.FREIGHT > 50 AND O.EMPLOYEEID = 5
AND O.REQUIREDDATE < '5/1/1997'
OR
O.FREIGHT>50 AND O.CUSTOMERID='VINET' AND O.REQUIREDDATE <
'5/1/1997' OR
O.REQUIREDDATE < '5/1/1997' OR O.FREIGHT > 50
```

As it can be seen in the above query the or clause is repeated and hence our application identifies the problem and transforms the query as

```
USE Northwind
SELECT ORDERID FROM ORDERS
WHERE   ORDERS.FREIGHT >50 OR ORDERS.REQUIREDDATE <'5/1/1997'
```

Now let us check one more type of queries which is a special case and is not done for all type of queries and can be called as an example of cost based optimization

```
USE Northwind
SELECT *
FROM EMPLOYEES
WHERE COUNTRY IN (
SELECT ORDERS.SHIPCOUNTRY
FROM ORDERS WHERE ORDERS.ORDERID = 10272
AND ORDERS.FREIGHT > 90)
```

The optimized query can be the original query or the following query

```
USE Northwind
SELECT CUSTOMERID, EMPLOYEEID, FREIGHT, ORDERDATE, ORDERID,
REQUIREDDATE, SHIPADDRESS, SHIPCITY, SHIPCOUNTRY, SHIPNAME,
SHIPPEDDATE, SHIPPOSTALCODE, SHIPREGION, SHIPVIA
FROM EMPLOYEES, ORDERS
WHERE COUNTRY = ORDERS.SHIPCOUNTRY
AND ORDERS.ORDERID = 10272 AND ORDERS.FREIGHT > 90
```

The application does this only if the nnumber of row in th inner table are large else the original query is displayed as optimized query

An SQL query to count the number of rows in a table

```
USE Northwind
SELECT COUNT(*)
FROM ORDERS
```

In the above query the astrick '*' is parsed as wildcard meaning all the column names now all column names are counted irrespective of them having the null property set to allowed or not and the the one having largers number of rows is returned.

The application changes this type of query as follows

```
USE Northwind
SELECT COUNT(ORDERID)
FROM ORDERS
```

Where orderid is thr primary key of the table ORDERS now the number of rows in the primary key column is equal to number of rows in the table.

The seceond query will take less time of execution

The Application merges two or more sets of data resulting from two or more queries using UNION. For example:

```
SELECT O.ORDERID
FROM ORDERS O
WHERE O.FREIGHT>50
UNION
SELECT O.ORDERID
FROM ORDERS O
WHERE O.REQUIREDDATE<'5/1/1997'
```

This same query can be rewritten, like the following example, and when doing so, performance will be boosted:

```
SELECT O.ORDERID
FROM ORDERS O
WHERE O.FREIGHT > 50 OR O.REQUIREDDATE < '5/1/1997'
```

Query optimization is a key ingredient for achieving good performance and simplifying administration. Our Tool query optimizer provides a tremendous breadth of capabilities. It incorporates wide variety of SQL transformation and innovative techniques for adjustments during query execution. Designing effective and correct SQL transformations is hard, developing a robust cost metric is elusive, and building extensible enumeration architecture is a significant undertaking

Query Processor

This module actually executes the query. The result of the query is shown in a separate form.

Predictive Word Capabilities

This module automatically detects and corrects misspelled words. This functionality extends only to key words and table names. There exists another limitation that is, the misspelled word must contain the same number of characters as the original word and also that only if there is a mismatch of three or less characters the word is corrected. As there could be a variety of avoidable errors in writing a particular word, we will provide a list below of the errors which this module is capable of correcting.

Documenting the Phase Efforts

After the completion of this phase we had come to a conclusion about all the design issues. We had decided upon what all features would be implemented and how we should go about the implementation. And hence we moved on to our next phase which consisted of coding and testing.

Chapter 7

Development Phase

Objective

The objective of the Development Phase was to convert the deliverable of the Design Phase into a complete information system. Much of the activity in the Development Phase addresses the computer programs that make up the system. This phase also puts in place the software, and communications environment for the system and other important elements of the overall systems.

The activities of this phase were mainly that of coding after the design phase. The activities of this phase translate the system design produced in the Design Phase into a working information system capable of addressing the information system requirement. The development phase contains activities for building the system, resting the system, and conducting Functional Qualification testing, to ensure the system functional processes satisfy the functional process requirements in the Functional Requirements Document. At the end of this phase, the system will be ready for the activities of the Integration and Testing Phase.

Tasks and Activities

Coding

A coding is a software engineering activity in which the developers must work to bring the system solution in a run able state. Coding activity was modularized based on the modules defined by us for the proposed system. These modules were divided into sub modules.

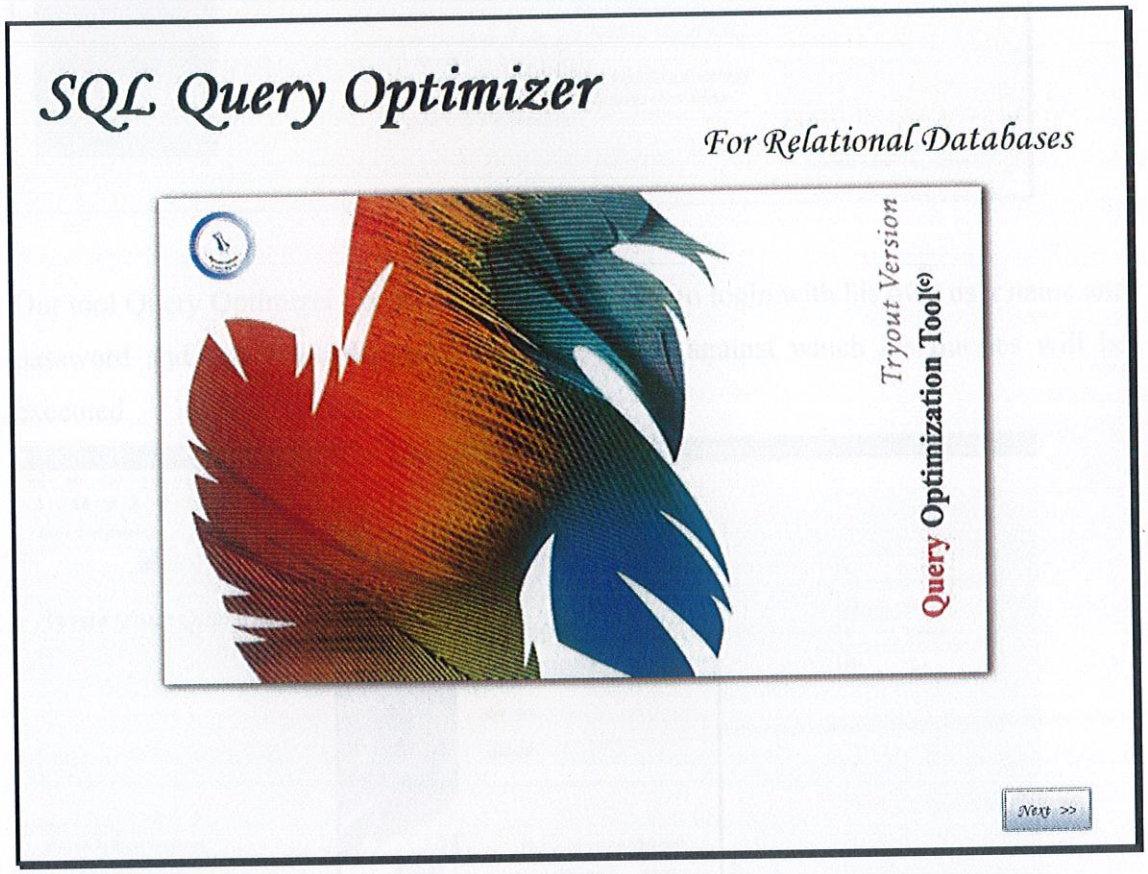
All the coding activity was divided among the team members in the form of modules and sub-modules and all of the members must work towards a common goal. All the interfaces were designed in this phase.

As the coding for the particular components of the module or the sub module gets completed they will be tested to see whether they are running in the desired manner as per the Functional requirement Document.

Tool Design

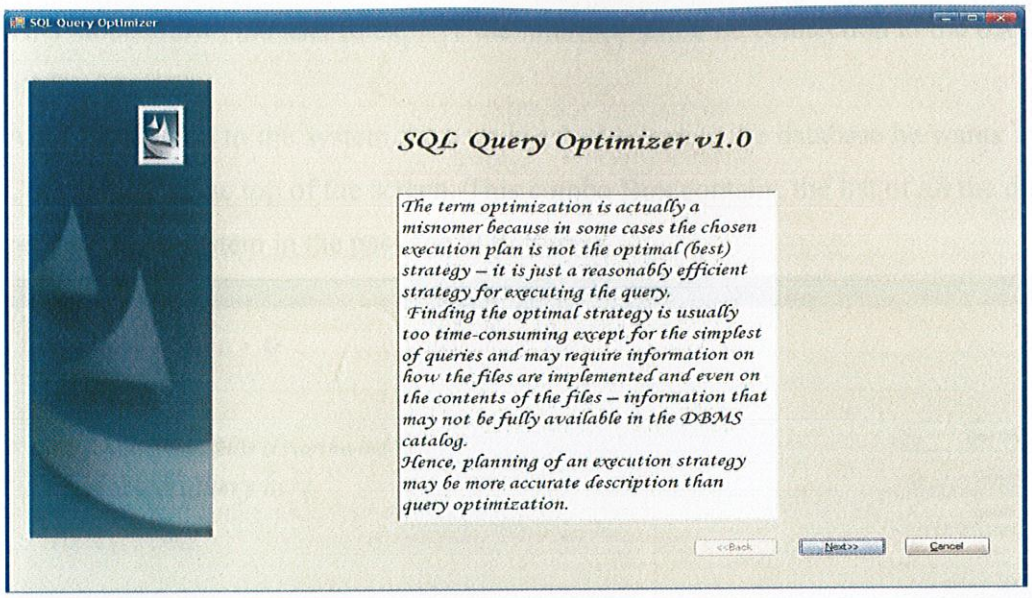
The system was designed and a running inter face was prepared

The front end was coded using VB.NET

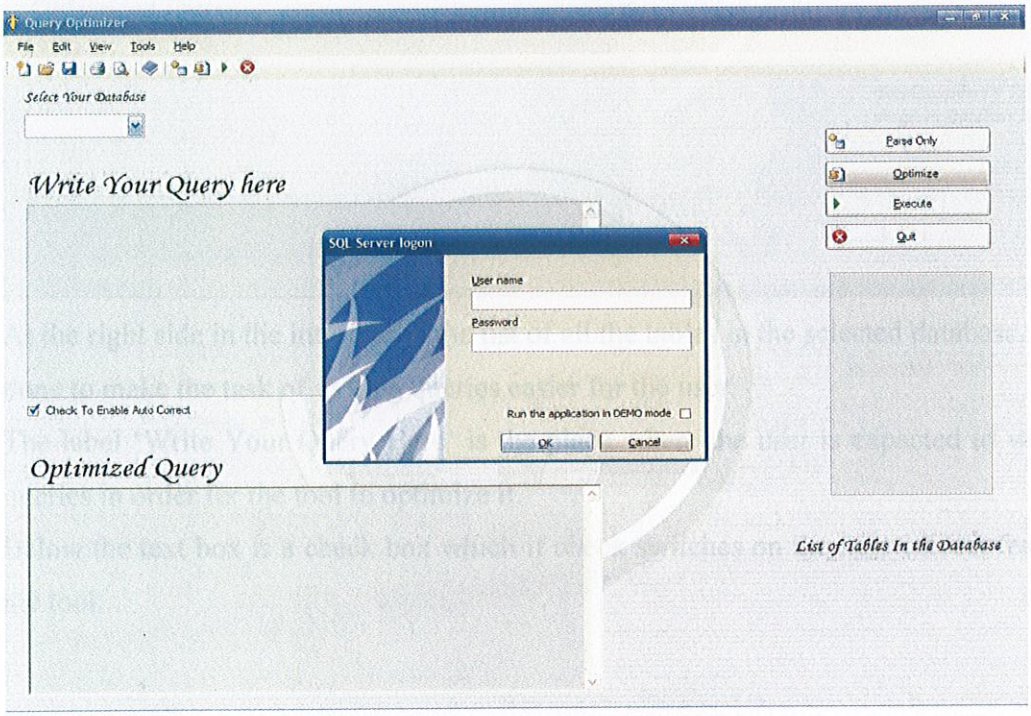


This screen is our welcome screen which is displayed while the loading is on.

After the loading the next screen is the information screen it gives a brief overview about the tool and gives an option for the user to exit if he wants



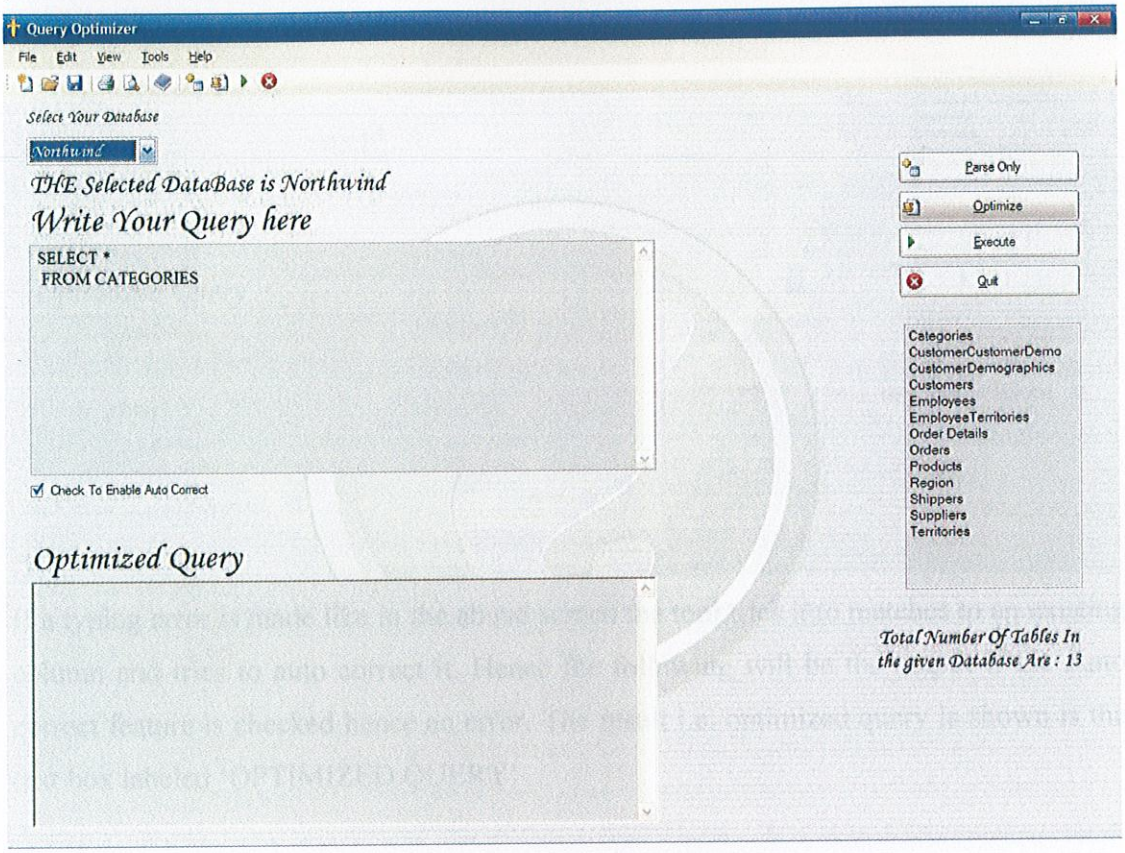
Our tool Query Optimizer gives an option to the user to login with his own user name and password and select the database which he wants against which his queries will be executed



This is the login Screen. The user is expected to login or he can chose the DEMO mode.

This mode allows the user to explore the interface while no connection to the data base is made.

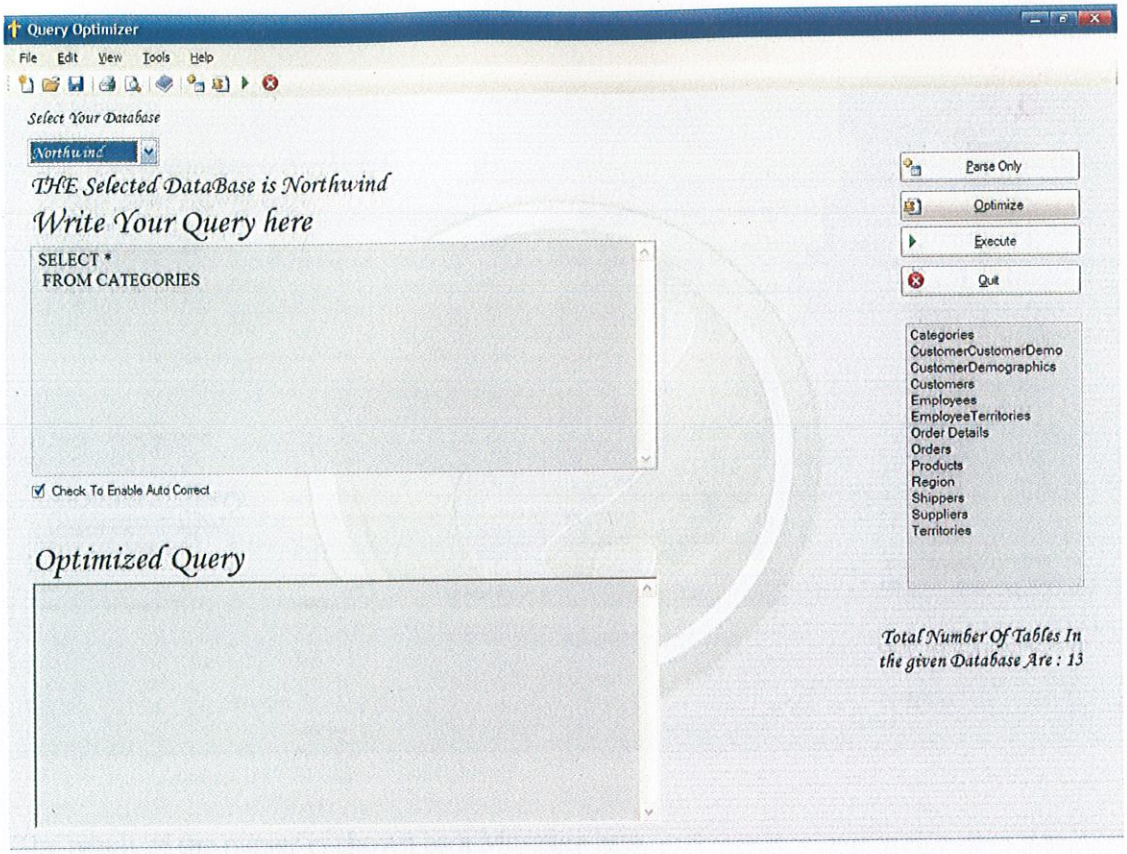
After logging on to the system the user is asked to select the database he wants from the Combo Box at the top of the screen. This combo Box contains the list of all the data base present in the system in the package SQL Server.



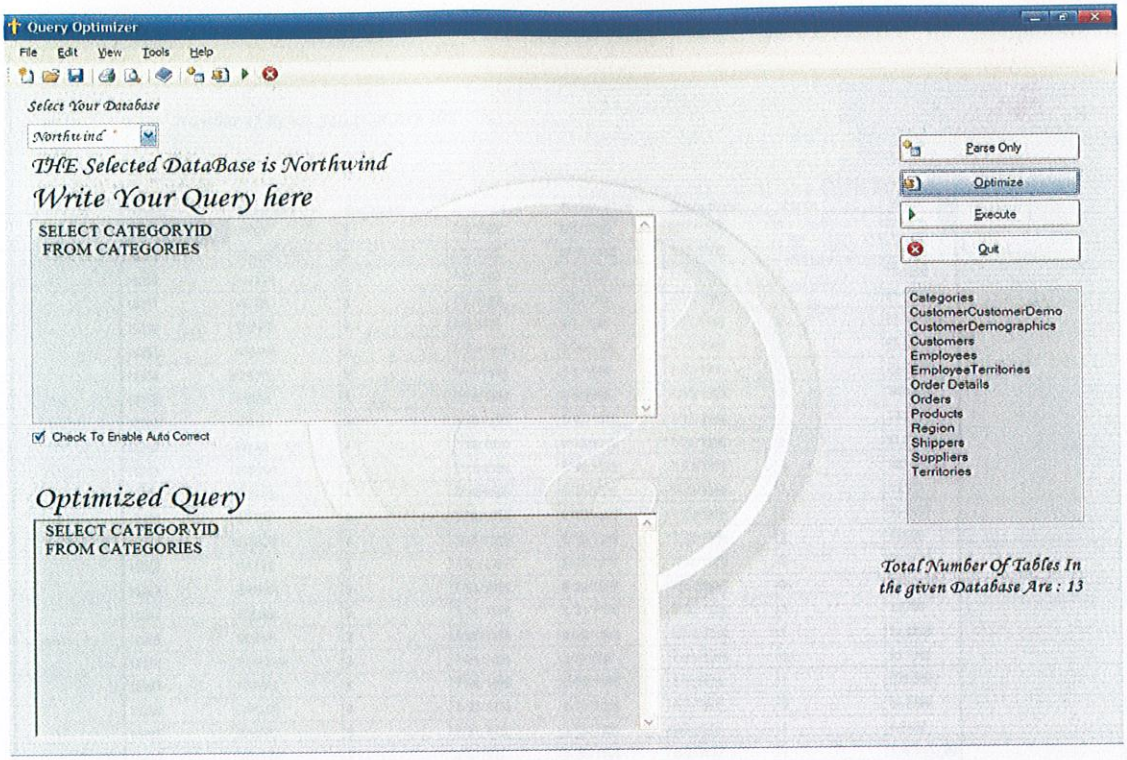
At the right side in the interface is the list of all the tables in the selected database. This is done to make the task of writing queries easier for the user.

The label 'Write Your Query Here' is the place where the user is expected to write his queries in order for the tool to optimize it.

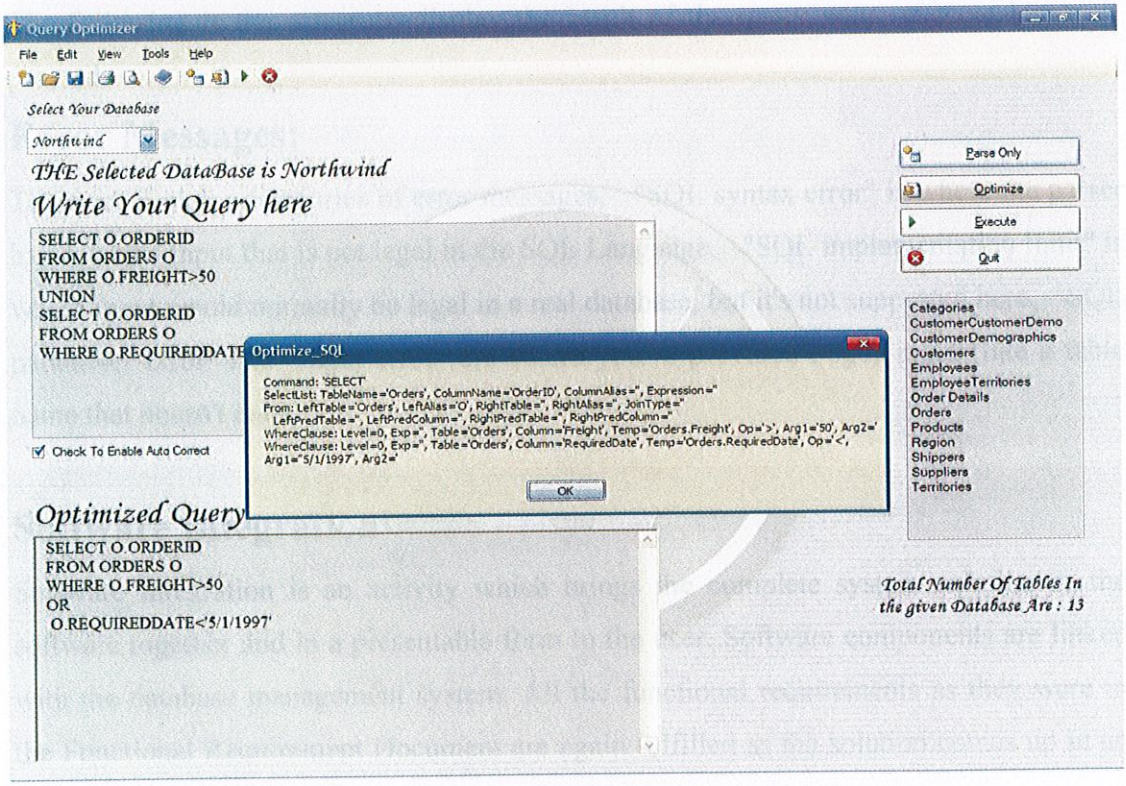
Below the text box is a check box which if check switches on the auto correct feature of the tool.

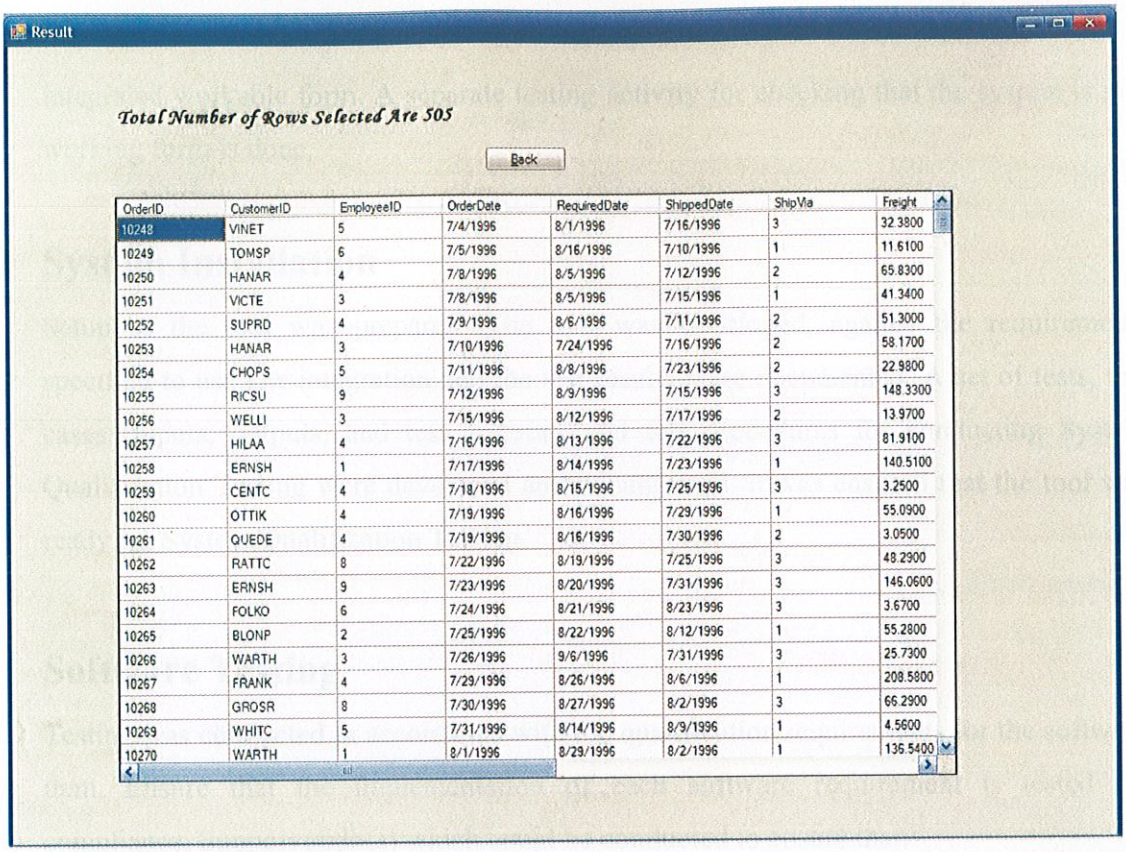


If a typing error is made like in the above screen the tool tries it to matches to an existing column and tries to auto correct it. Hence the following will be the output if the Auto correct feature is checked hence an error. The result i.e. optimized query is shown is the text box labeled 'OPTIMIZED QUERY'



The result of the parser is shown as a Message box.





OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight
10248	VINET	5	7/4/1996	8/1/1996	7/16/1996	3	32.3800
10249	TOMSP	6	7/5/1996	8/16/1996	7/10/1996	1	11.6100
10250	HANAR	4	7/8/1996	8/5/1996	7/12/1996	2	65.8300
10251	VICTE	3	7/8/1996	8/5/1996	7/15/1996	1	41.3400
10252	SUPRD	4	7/9/1996	8/6/1996	7/11/1996	2	51.3000
10253	HANAR	3	7/10/1996	7/24/1996	7/16/1996	2	58.1700
10254	CHOPS	5	7/11/1996	8/8/1996	7/23/1996	2	22.9800
10255	RICSU	9	7/12/1996	8/9/1996	7/15/1996	3	148.3300
10256	WELLI	3	7/15/1996	8/12/1996	7/17/1996	2	13.9700
10257	HILAA	4	7/16/1996	8/13/1996	7/22/1996	3	81.9100
10258	ERNSH	1	7/17/1996	8/14/1996	7/23/1996	1	140.5100
10259	CENTC	4	7/18/1996	8/15/1996	7/25/1996	3	3.2500
10260	OTTIK	4	7/19/1996	8/16/1996	7/29/1996	1	55.0900
10261	QUEDE	4	7/19/1996	8/16/1996	7/30/1996	2	3.0500
10262	RATTC	8	7/22/1996	8/19/1996	7/25/1996	3	48.2900
10263	ERNSH	9	7/23/1996	8/20/1996	7/31/1996	3	146.0600
10264	FOLKO	6	7/24/1996	8/21/1996	8/23/1996	3	3.6700
10265	BLONP	2	7/25/1996	8/22/1996	8/12/1996	1	55.2800
10266	WARTH	3	7/26/1996	9/6/1996	7/31/1996	3	25.7300
10267	FRANK	4	7/29/1996	8/26/1996	8/6/1996	1	208.5800
10268	GROSR	8	7/30/1996	8/27/1996	8/2/1996	3	66.2900
10269	WHITC	5	7/31/1996	8/14/1996	8/9/1996	1	4.5600
10270	WARTH	1	8/1/1996	8/29/1996	8/2/1996	1	136.5400

The following is the screen to display the result of the query when execute button is pressed.

Error Messages:

There are 3 major categories of error messages. "SQL syntax error" is where the parser has detected input that is not legal in the SQL Language. "SQL implementation limit" is where input would normally be legal in a real database, but it's not supported here. "SQL parameter error" and "Input error" are where you've provided bogus input (like a table name that doesn't exist).

Software Integration

Software Integration is an activity which brings the complete system solution or the software together and in a presentable form to the user. Software components are linked with the database management system. All the functional requirements as they were in the Functional Requirement Document are again fulfilled as the solution comes up in an

integrated workable form. A separate testing activity for checking that the system is in a working form is done.

System Installation

Setup of the tool was prepared .The tool was developed, against the requirements specified to us. The integration and the test results were documented. A set of tests, test cases (inputs, outputs, and test criteria), and test procedures for conducting System Qualification Testing were developed and documented. It was ensured that the tool was ready for System Qualification Testing.

Software Testing

Testing was conducted in accordance with the qualification requirements for the software item. Ensure that the implementation of each software requirement is tested for compliance. Support audit(s) which could be conducted to ensure that:

- Coded software tool reflect the design documentation.
- The acceptance review and testing requirement prescribed by the documentation are adequate for the acceptance of the software tool.
- Test data comply with the specification.
- Software tool was successfully tested and meet its specifications.
- Test reports are correct and discrepancies between actual and expected results have been resolved.

The results of the audits were documented. We also, establish a baseline for the design and code of the software item.

Documenting the Phase Efforts

After the completion of this phase we had developed running software. The tool had the look that we wanted to give. All the requirements were met that were specified. We at the end of this phase had developed running software. The software was developed with all the specification met and all the modules coded. The coding was done and the executable files were created. With the end of this phase we marked

Limitations

A large number of Heuristic query transformations exist but the list is huge and so only a few could be implemented in our project

Only one type of dynamic or non static transformation is applied.

This is a demonstration project and not a commercial one. It has been tested but not thoroughly tested in a production environment.

Conclusion

Many factors determine performance of a query

- ✓ Query Processing Engine
- ✓ Query Optimizer
- ✓ Physical database design

Understanding of all factors is necessary to boost performance. Optimization is much more than transformations and query equivalence. The infrastructure for optimization is significant. Designing effective and correct SQL transformations is hard, developing a robust cost metric is elusive, and building extensible enumeration architecture is a significant undertaking. Despite many years of work, significant open problems remain. However, we, with our tool, have tried to develop an understanding of the existing engineering framework which is necessary for making effective contribution to the area of query optimization.

Our application **Query Optimization in Database Systems** aimed at providing an insight to Query optimization. As the field is relatively less explored, a lot can be done in the field.

To the best of our knowledge, the proposed Query optimization Tool is a unique tool based on statistical information about the resources contained in the underlying ontology. As the evaluation shows, the approach seems to be reasonable and we believe this is the way to go. Obviously, more research work is required to get even more accurate estimations. It is remarkable that a few optimization rules which aim at the common goal of minimizing the intermediate result set and in turn the query execution time, highly affect the performance of a query. The optimization work discussed in this project, focuses on static query reordering in order to get an execution plan which is optimal according to heuristics. Static optimization techniques may also be combined with dynamic techniques to achieve better results especially when static techniques do not lead to any effective optimization.

Like any software, our tool can be improved by adding several database management functionalities such as creating plan tables which can be shown to the user.

Bibliography

Books:

Date, C. **An Introduction to Database Systems**, *Addison-Wesely Publishing Co.,*

Elmasri, R. And Navathe, **Fundamentals of Database Systems**, *Benjamin Cumings Co.*

Ramakrishnan Geherke **Database Management System** *Tata Mc Graw Hill.*

Paul N Weinberg **The Complete Reference SQL** *Tata Mc Graw Hill.*

Dan Tow **SQL Tunning** *O'Reilly publications.*

Stephane Faroult, **The Art of SQL** *O'Reilly publications.*

Thomas Connolly And Carolyn Begg, **Database Systems** *Pearson Educatioon.*

White Papers

Query Optimization in Oracle9*i*

An Oracle White Paper

February 2002

Oracle® Database

Performance Tuning Guide

10g Release 1 (10.1)

Research Papers

An Overview of Query Optimization in Relational Systems

Surajit Chaudhuri, Microsoft Research

Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems pages 34-43.

Estimating Compilation Time of a Query Optimizer

Ihab F. Ilyas

Jun Rao

Guy Lohman

SIGMOD Conference 2003: page 373-384

Query Optimization

Yannis E. Ioannidis

Computer Sciences Department

University of Wisconsin

ACM Computing Surveys, Vol. 30, No. 1, March 1999, page 121-123.

Design and Implementation of a Query Optimizer Analyzer

Mohammed Aslam

July 2006

Web

http://www.microsoft.com/sql/techinfo/productdoc/2000/books.asp
http://en.wikipedia.org
http://google.com
http://www.sql-server-performance.com/transact_sql_select.asp
http://www.sql-server-performance.com/tuning_joins.asp